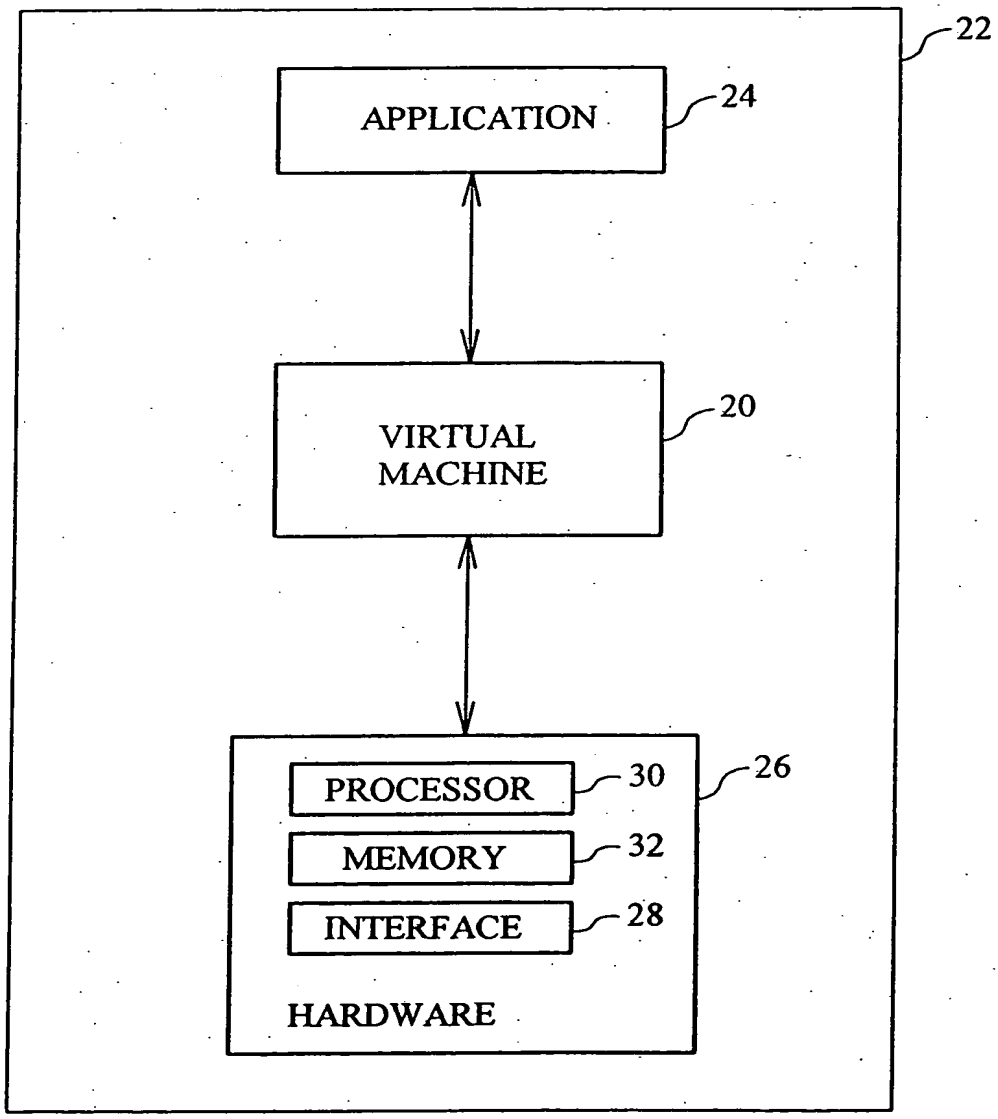
1 of 38

Fig. 1



Petitioner - Ex. 1003 - Page 38

Fig. 1A

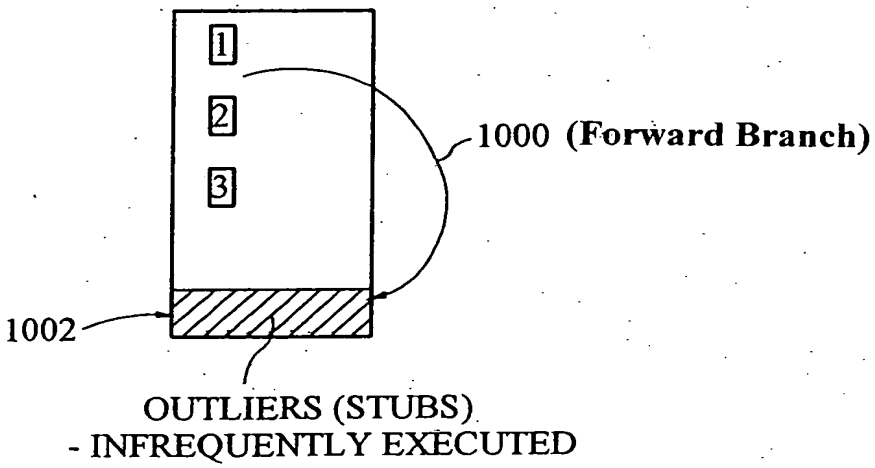
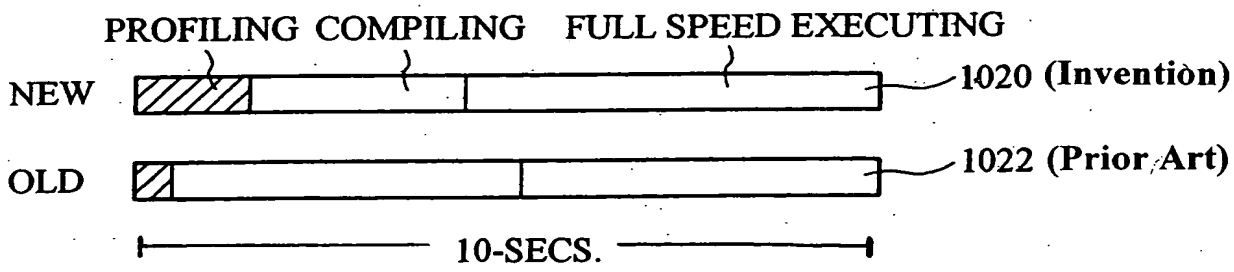

1
2
3
1000 (Forward Branch)
1002
OUTLIERS (STUBS)
- INFREQUENTLY EXECUTED

Fig. 1B


PROFILING
COMPILING
FULL SPEED EXECUTING
NEW
1020 (Invention)
OLD
1022 (Prior Art)
10-SECS.

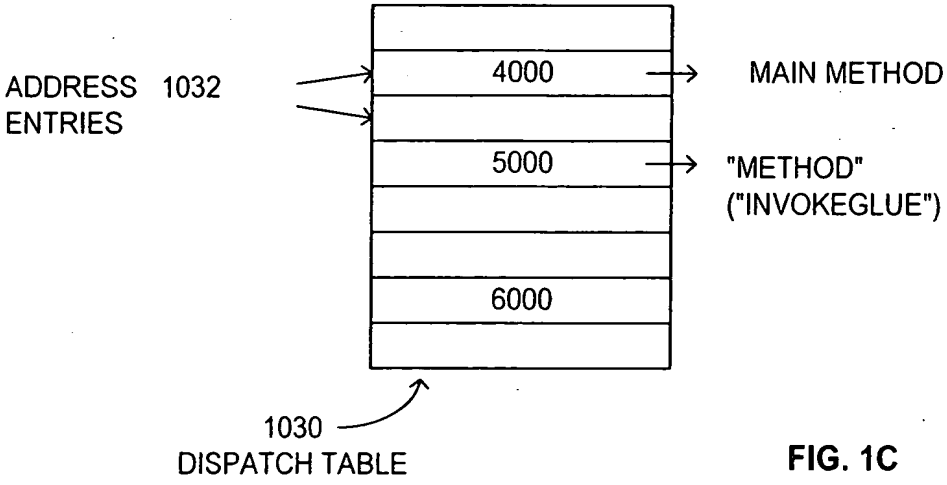

ADDRESS 1032
ENTRIES
4000
MAIN METHOD
5000
"METHOD"
("INVOKEGLUE")
6000
1030
DISPATCH TABLE

FIG. 1C

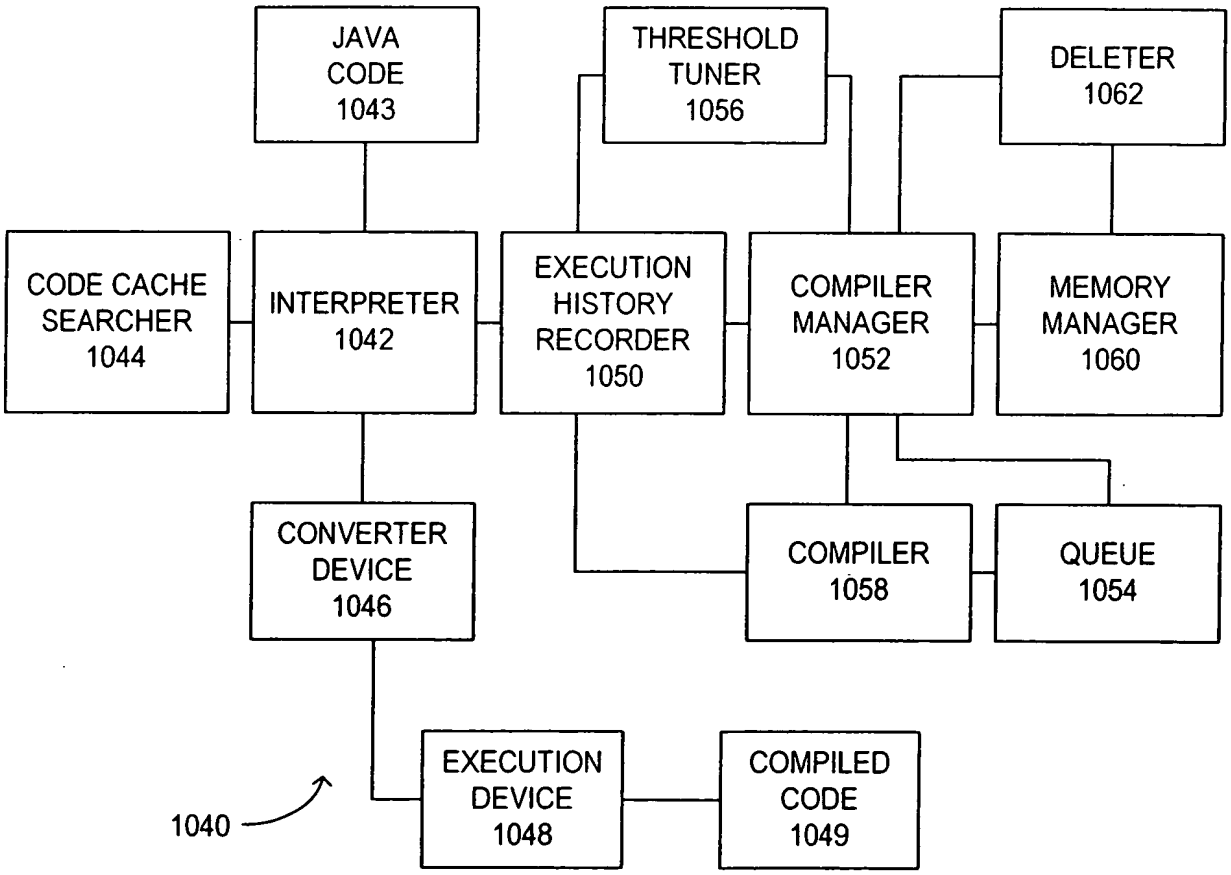

JAVA CODE 1043
THRESHOLD TUNER 1056
DELETER 1062
CODE CACHE SEARCHER 1044
INTERPRETER 1042
EXECUTION HISTORY RECORDER 1050
COMPILER MANAGER 1052
MEMORY MANAGER 1060
CONVERTER DEVICE 1046
COMPILER 1058
QUEUE 1054
1040
EXECUTION DEVICE 1048
COMPILED CODE 1049

FIG. 1D

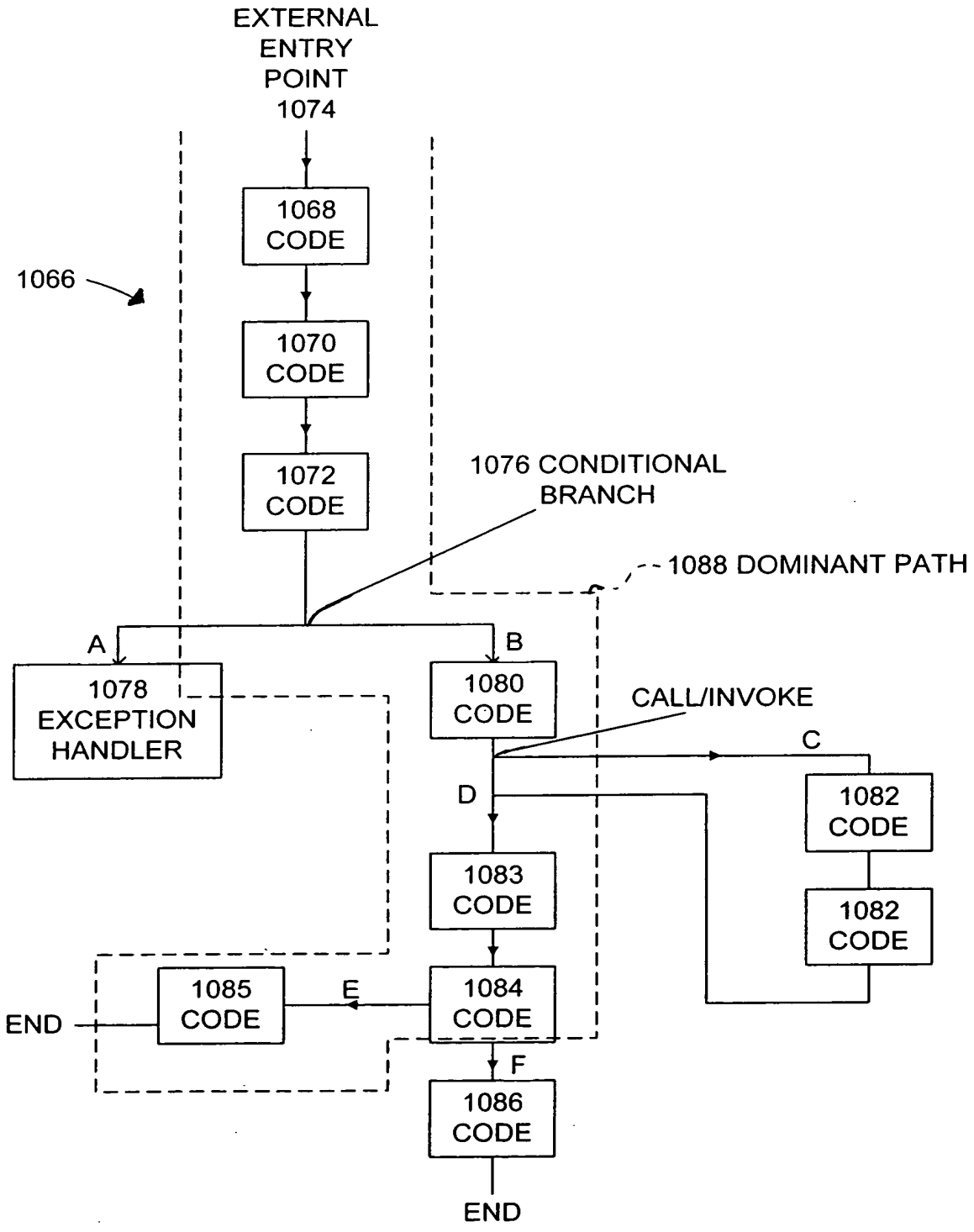

EXTERNAL ENTRY POINT 1074
1066
1068 CODE
1070 CODE
1072 CODE
1076 CONDITIONAL BRANCH
1088 DOMINANT PATH
A
B
1078 EXCEPTION HANDLER
1080 CODE
CALL/INVOKE
C
D
1082 CODE
1083 CODE
1082 CODE
1085 CODE
E
1084 CODE
END
F
1086 CODE
END

FIG. 1E

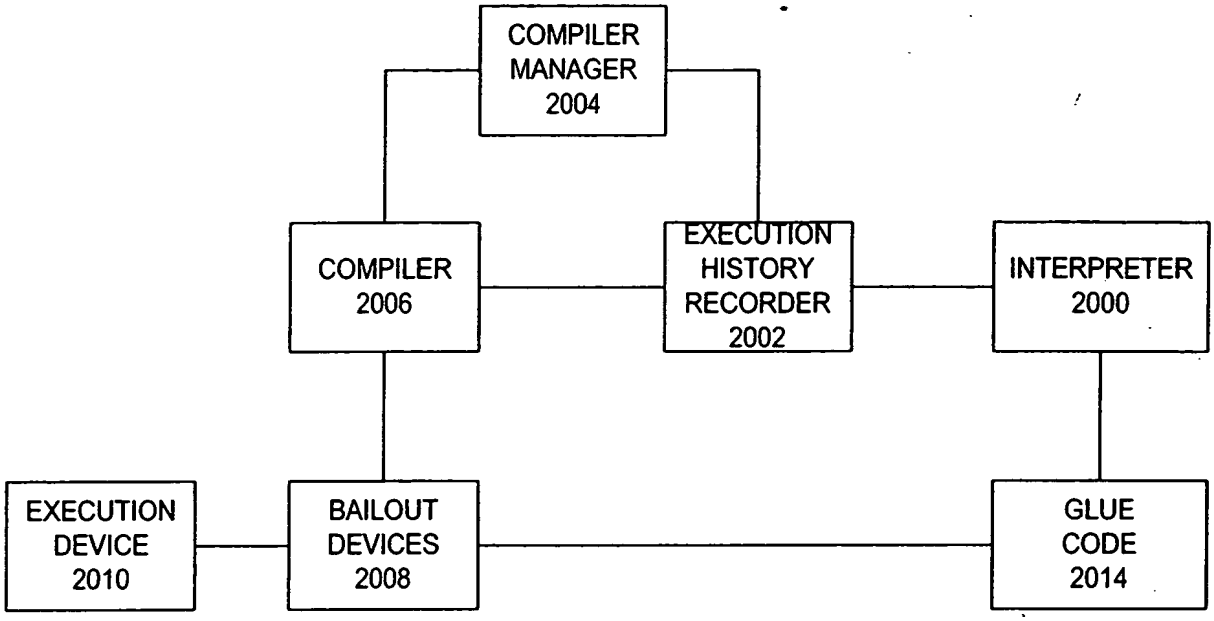


FIG. 2A

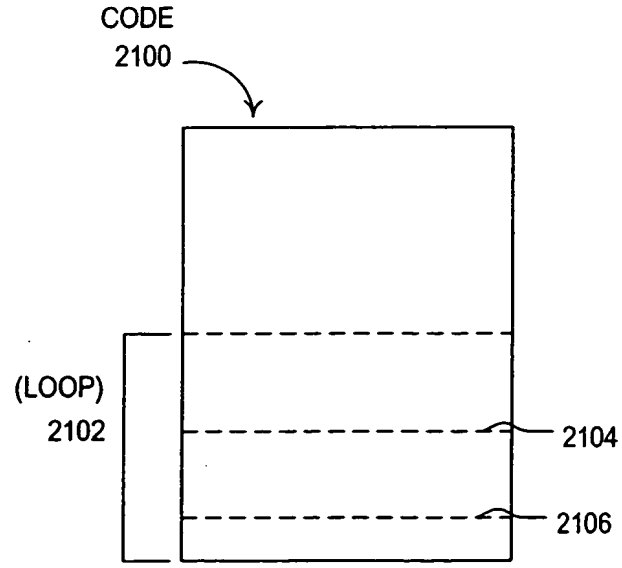


FIG. 2B

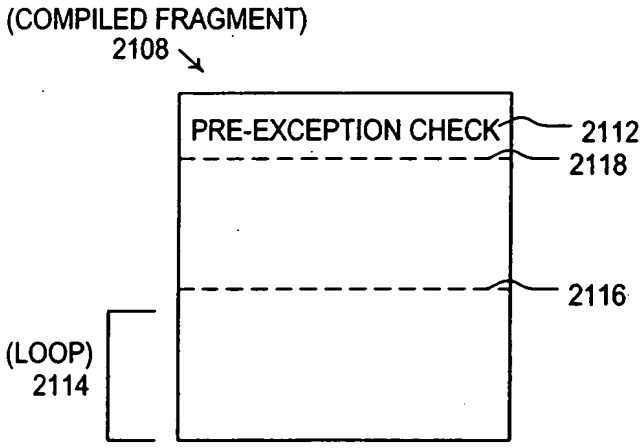


FIG. 2C

FIG. 3A

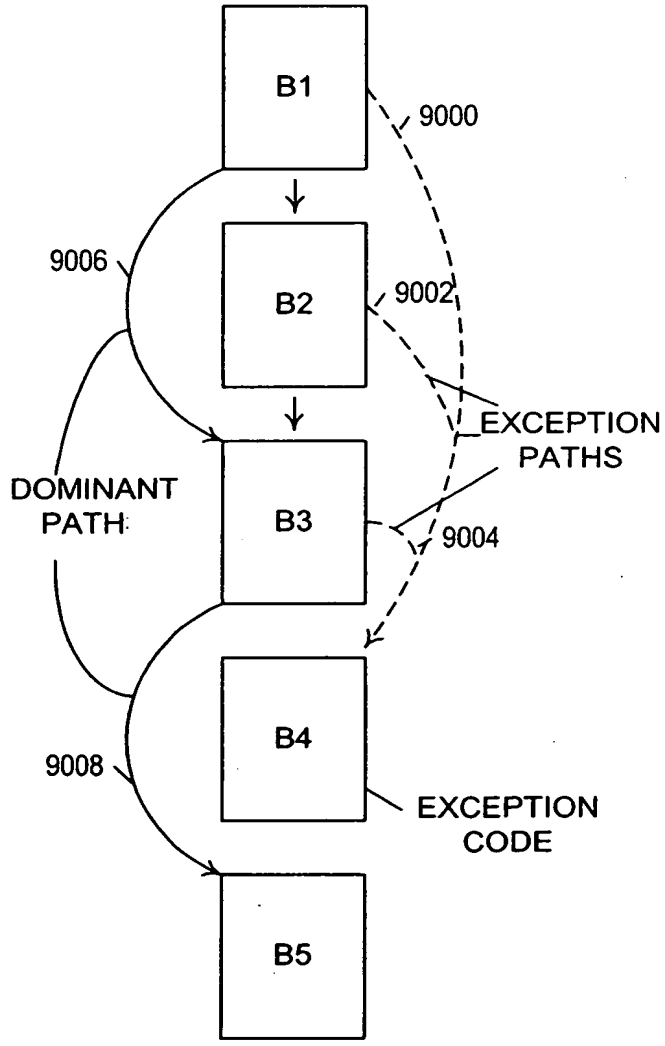


FIG. 3B

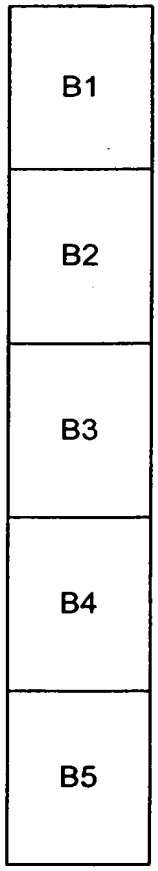
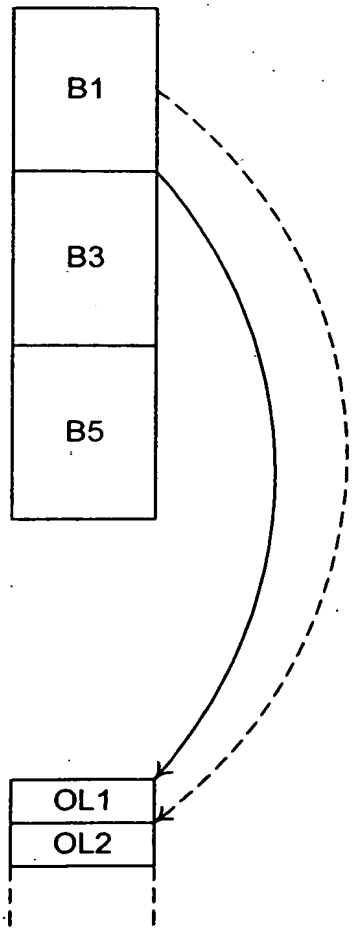


FIG. 3C



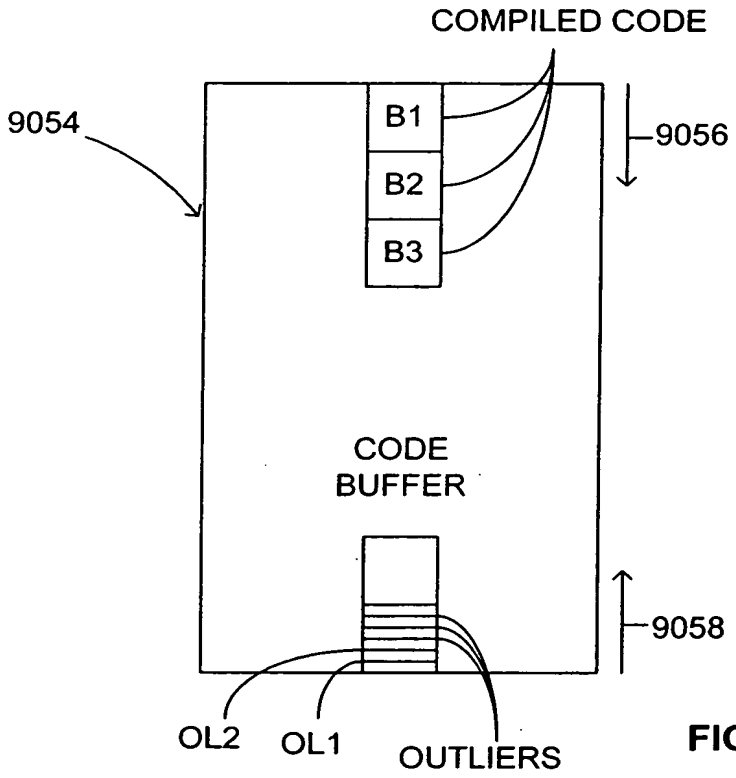
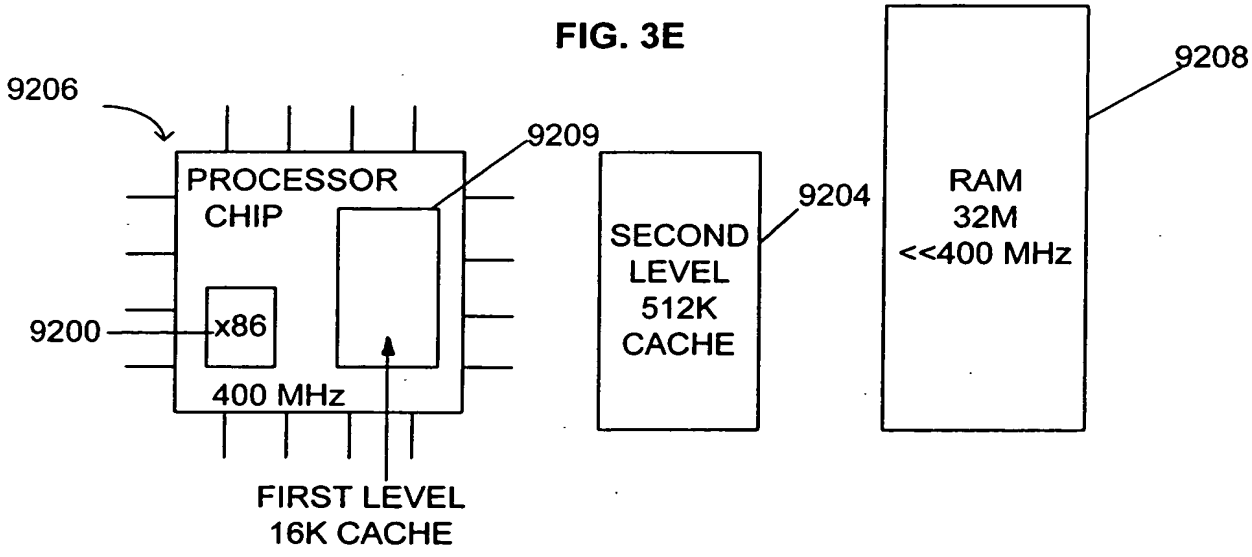

COMPILED CODE
9054
B1
B2
B3
9056
CODE
BUFFER
9058
OL2
OL1
OUTLIERS

FIG. 3D

FIG. 3E


9206
9209
PROCESSOR
CHIP
9200
x86
400 MHz
FIRST LEVEL
16K CACHE
SECOND
LEVEL
512K
CACHE
9204
RAM
32M
<<400 MHz
9208

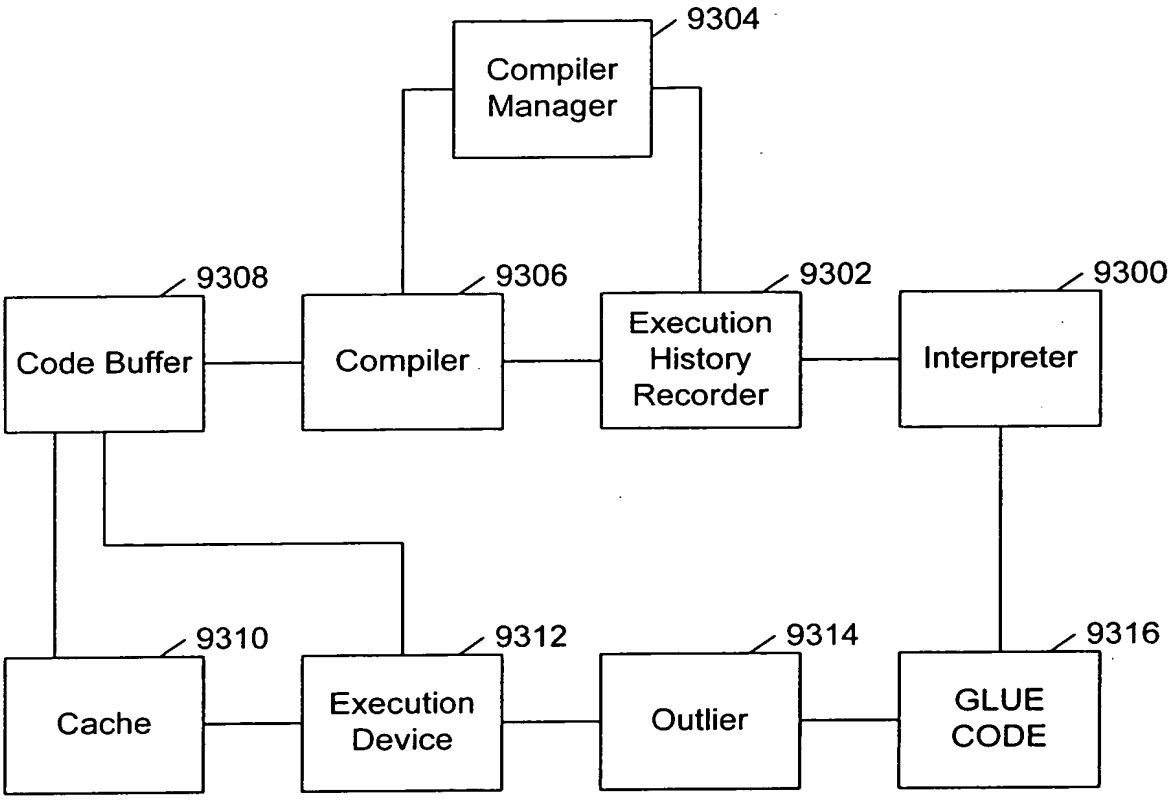

9304
Compiler Manager
9308
9306
9302
9300
Code Buffer
Compiler
Execution History Recorder
Interpreter
9310
9312
9314
9316
Cache
Execution Device
Outlier
GLUE CODE

FIG. 3F

Fig. 4A

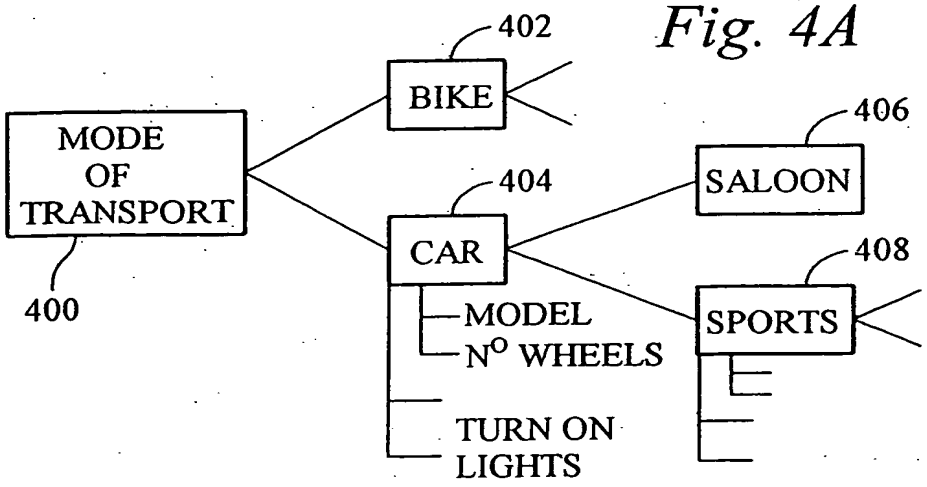


Fig. 4B

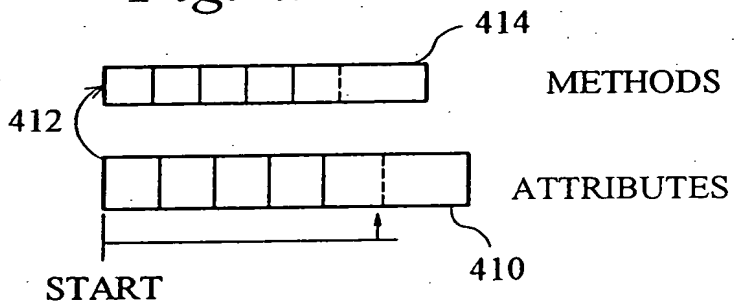
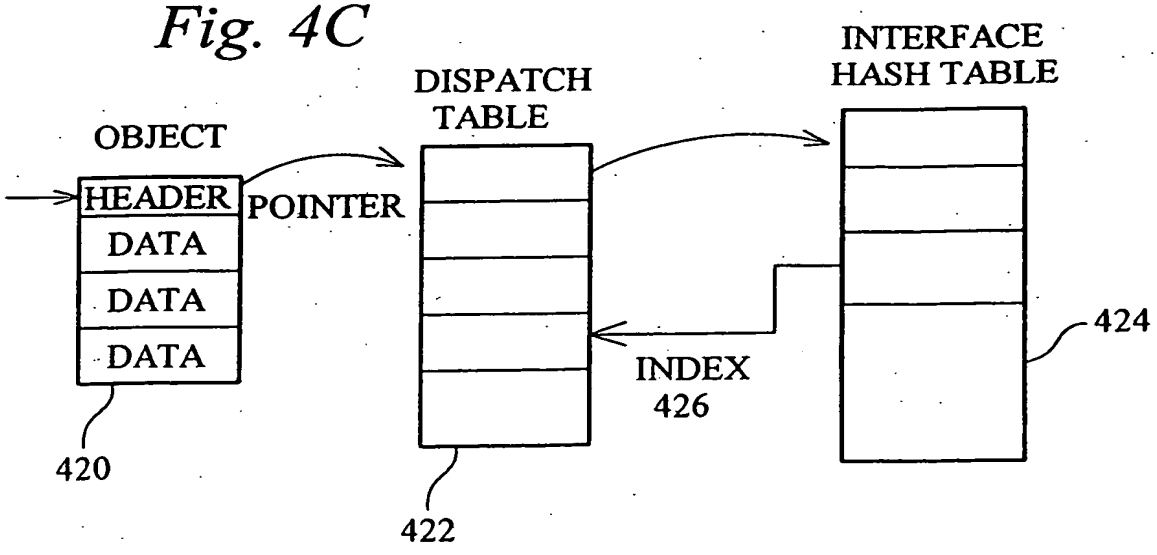
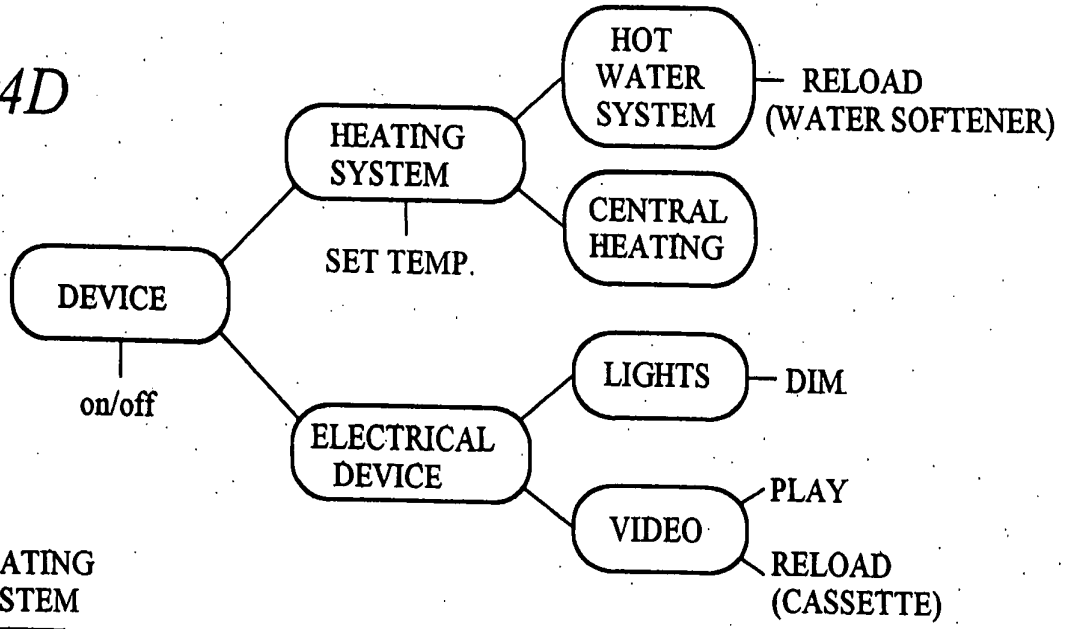


Fig. 4C



08585426.050104

Fig. 4D



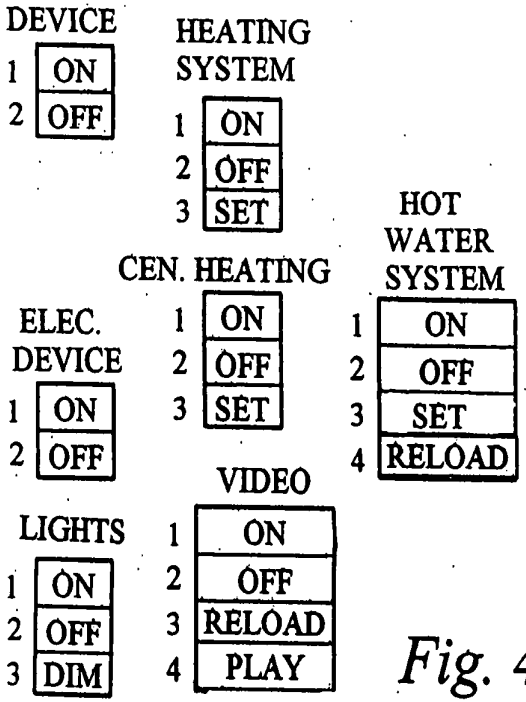
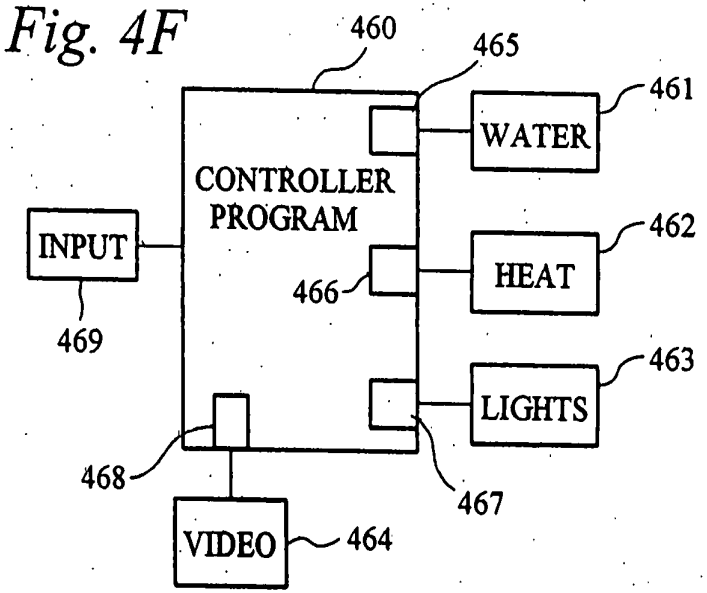


Fig. 4E

Fig. 4F



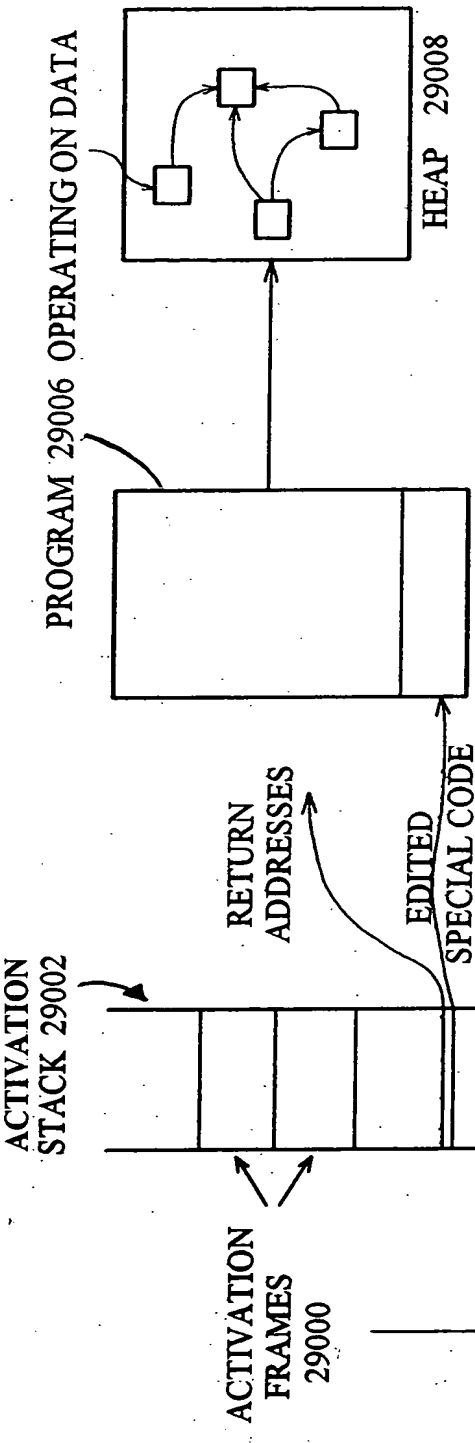

HEAP 29008
PROGRAM 29006 OPERATING ON DATA
EDITED SPECIAL CODE
RETURN ADDRESSES
ACTIVATION STACK 29002
ACTIVATION FRAMES 29000
29004
MOST RECENT ACTIVATION

Fig. 5A

Fig. 5B

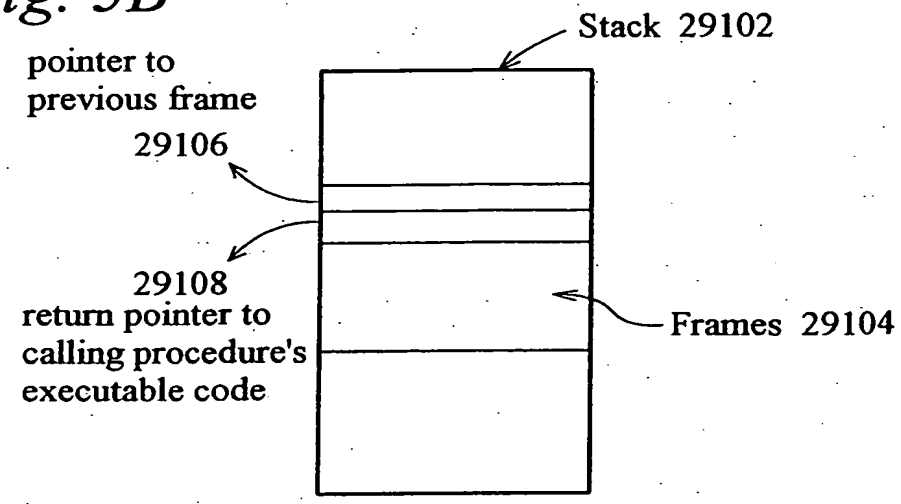


Fig. 5C

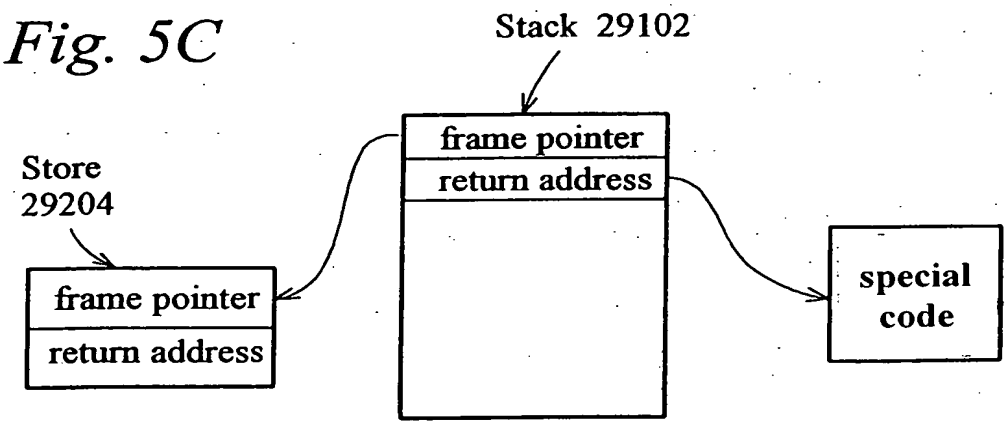
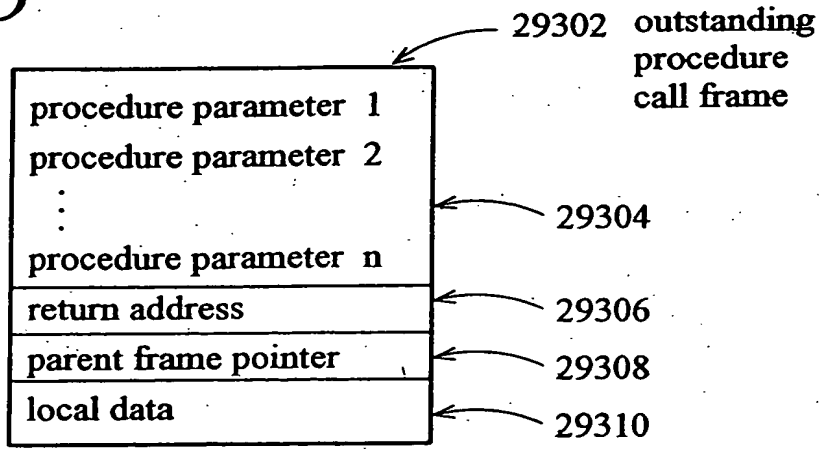


Fig. 5D



0058456-050501

Fig. 5E

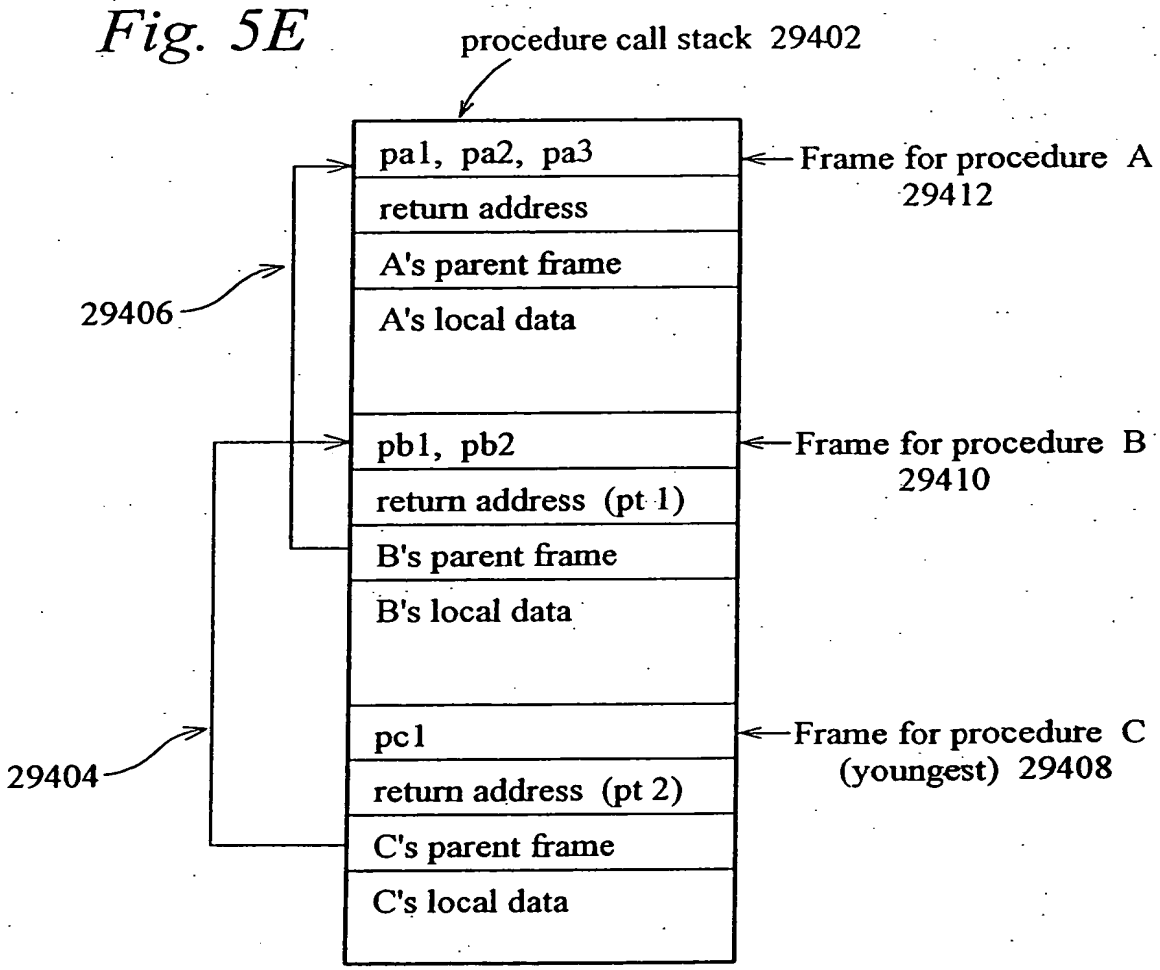
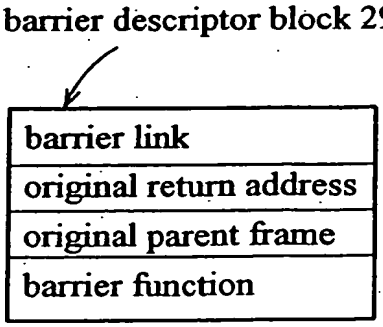

procedure call stack 29402
pa1, pa2, pa3
Frame for procedure A 29412
return address
A's parent frame
A's local data
29406
pb1, pb2
Frame for procedure B 29410
return address (pt 1)
B's parent frame
B's local data
pc1
Frame for procedure C (youngest) 29408
29404
return address (pt 2)
C's parent frame
C's local data

Fig. 5F


barrier descriptor block 29502
barrier link
original return address
original parent frame
barrier function

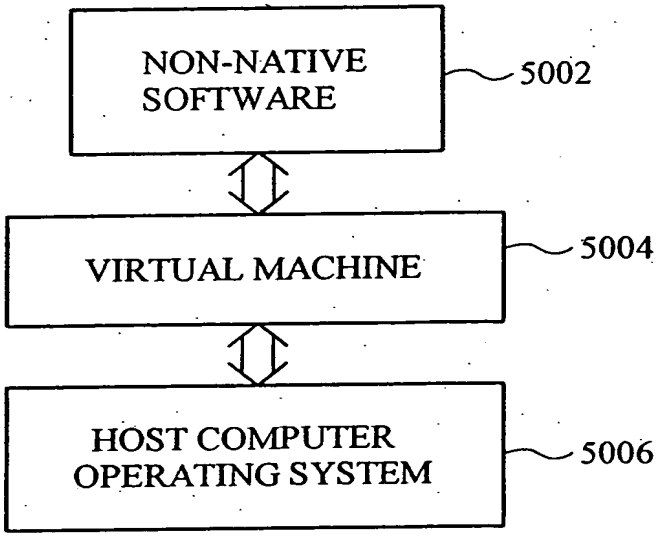


Fig. 6A

WO 2004/061660 PCT/US2003/041501

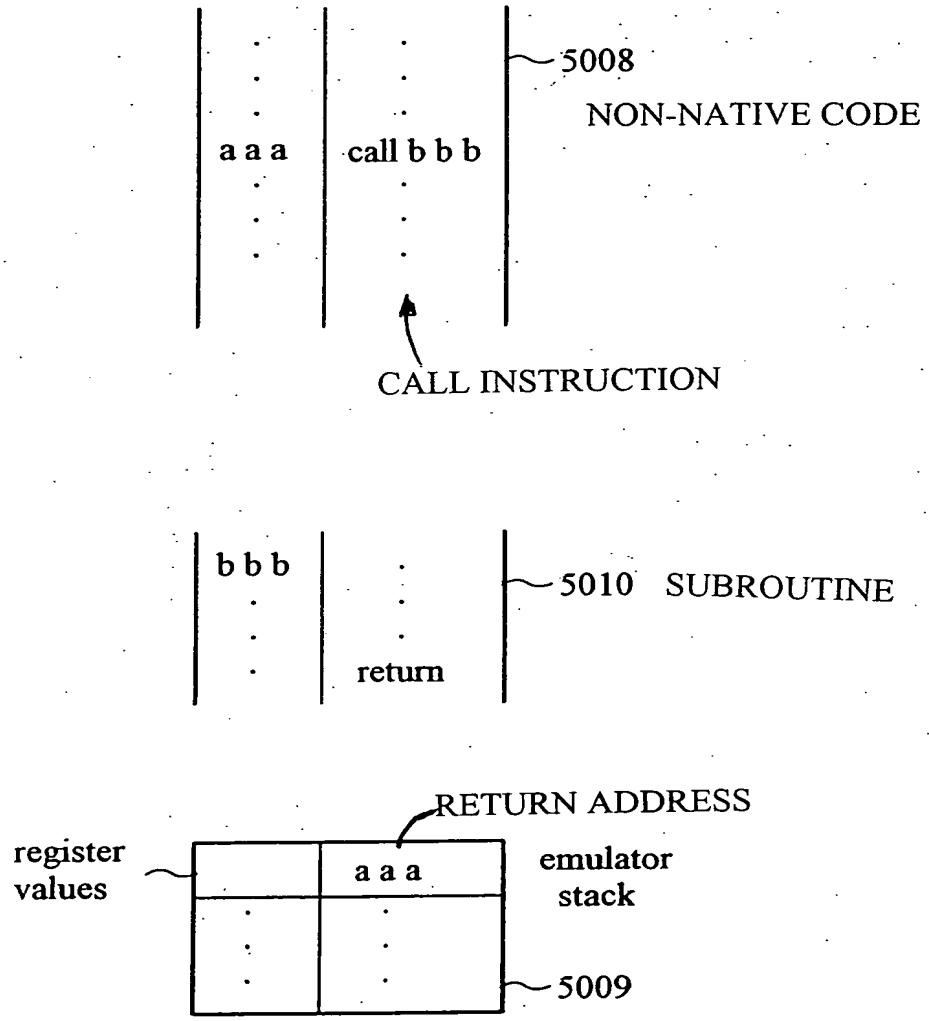


Fig. 6B

Petitioner's Exhibit 1005 - Page 1005

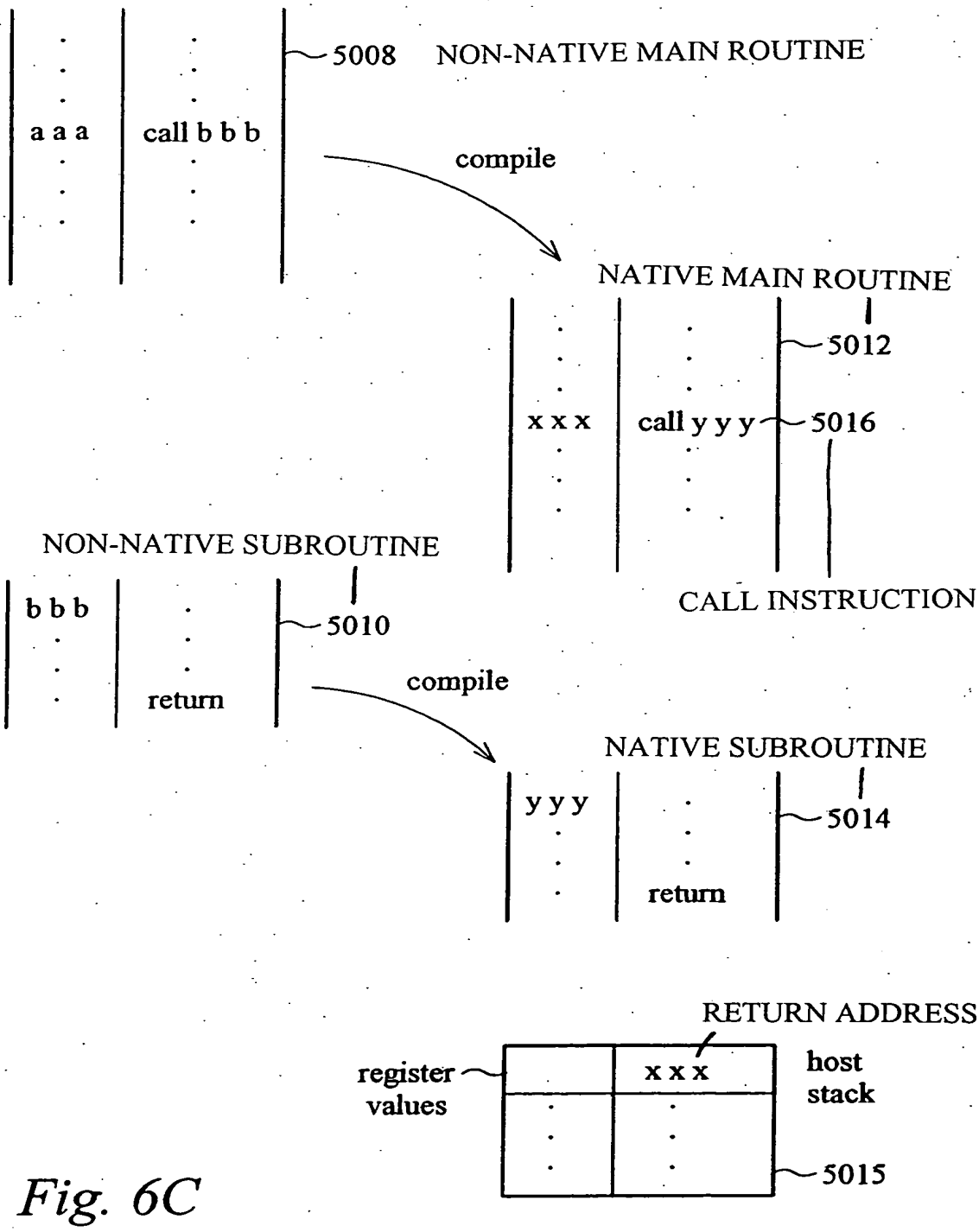
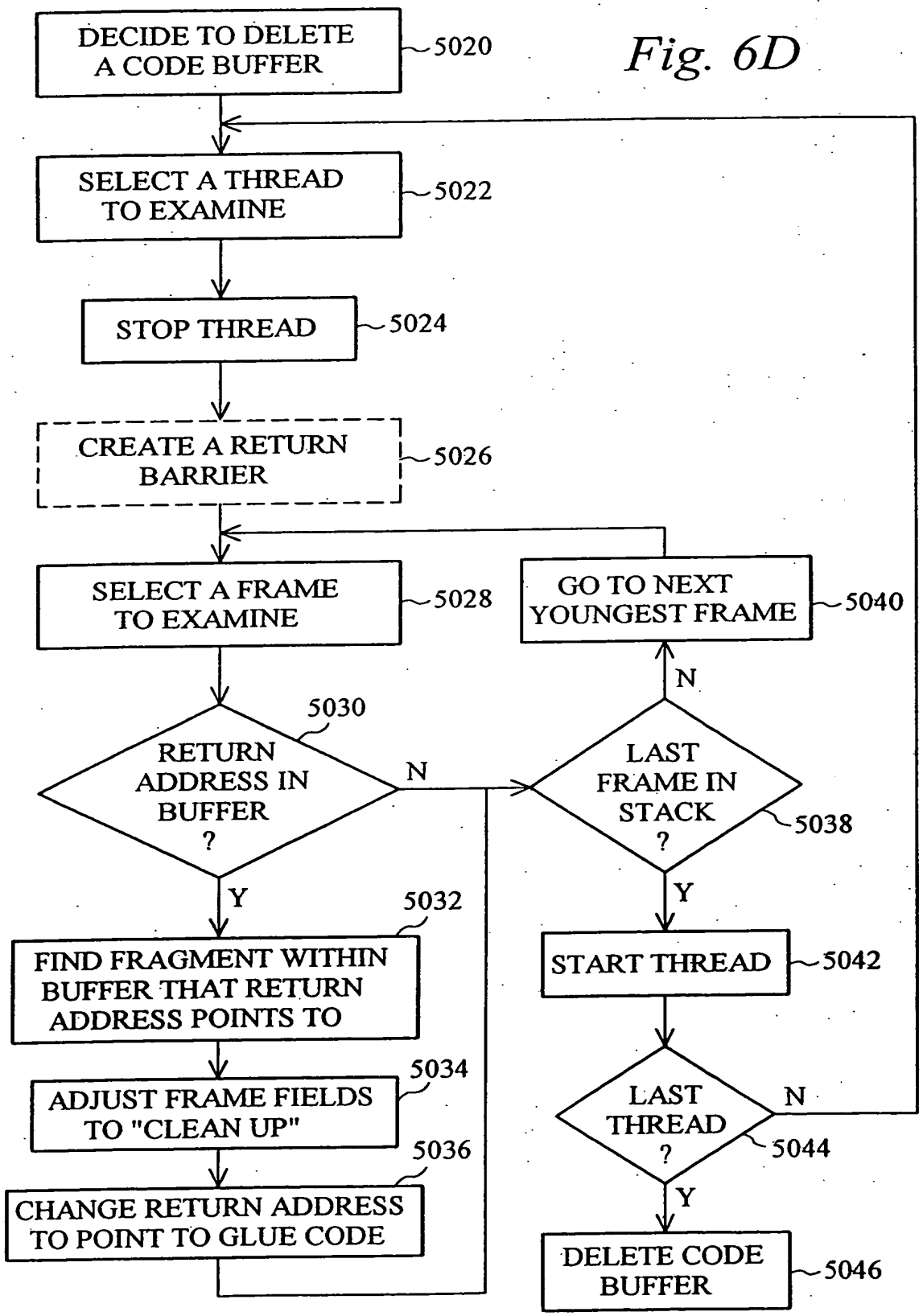

5008
NON-NATIVE MAIN ROUTINE
a a a
call b b b
compile
NATIVE MAIN ROUTINE
5012
x x x
call y y y
5016
NON-NATIVE SUBROUTINE
b b b
5010
return
CALL INSTRUCTION
compile
NATIVE SUBROUTINE
y y y
5014
return
RETURN ADDRESS
register values
x x x
host stack
5015

Fig. 6C

Fig. 6D



Docket No. 50T5601

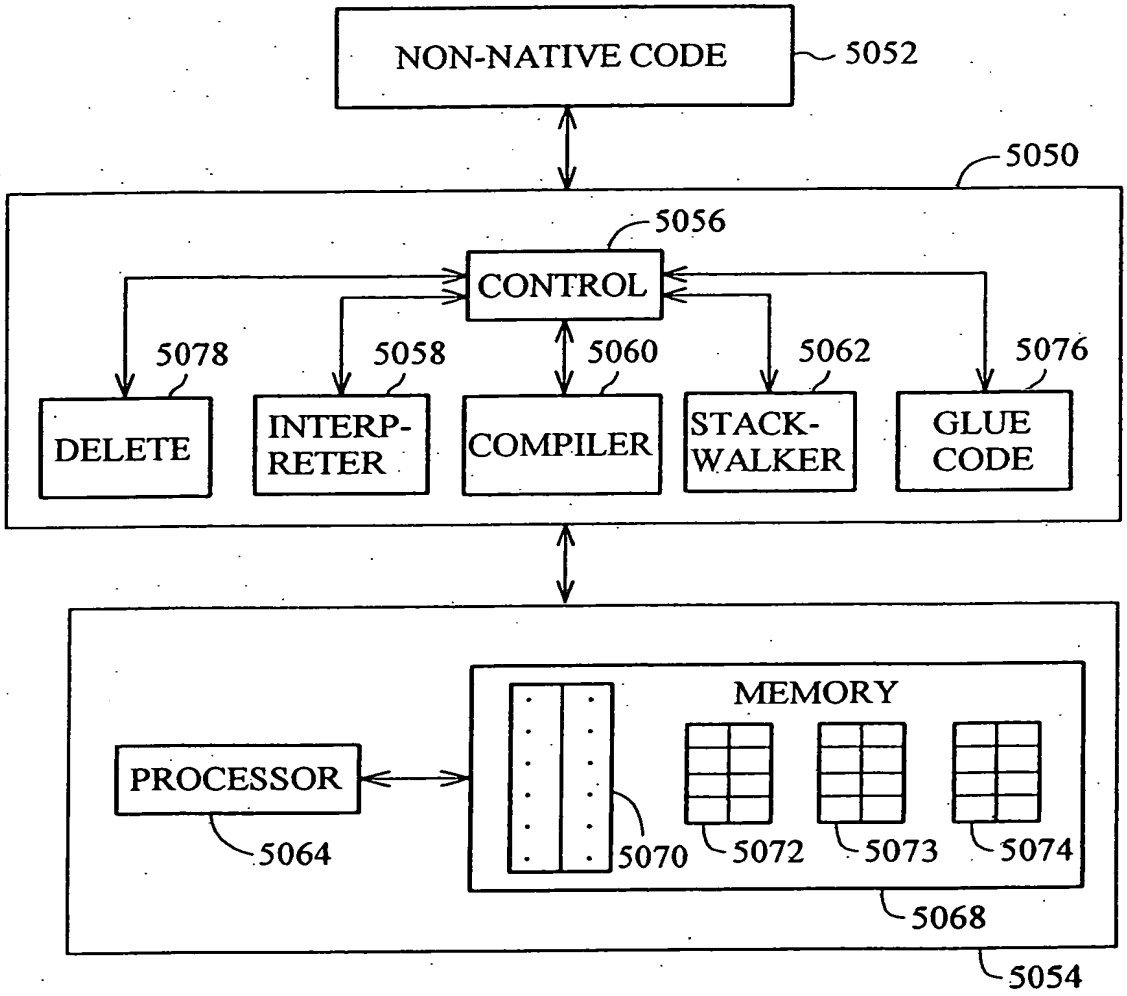

NON-NATIVE CODE
5052
5050
5056
CONTROL
5078
5058
5060
5062
5076
DELETE
INTERP-RETER
COMPILER
STACK-WALKER
GLUE CODE
MEMORY
PROCESSOR
5064
5070
5072
5073
5074
5068
5054

Fig. 6E

Fig. 7A

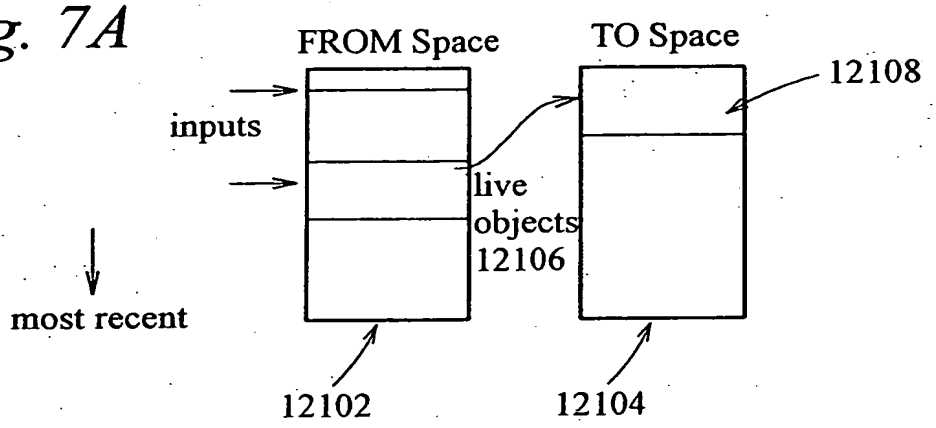


Fig. 7B

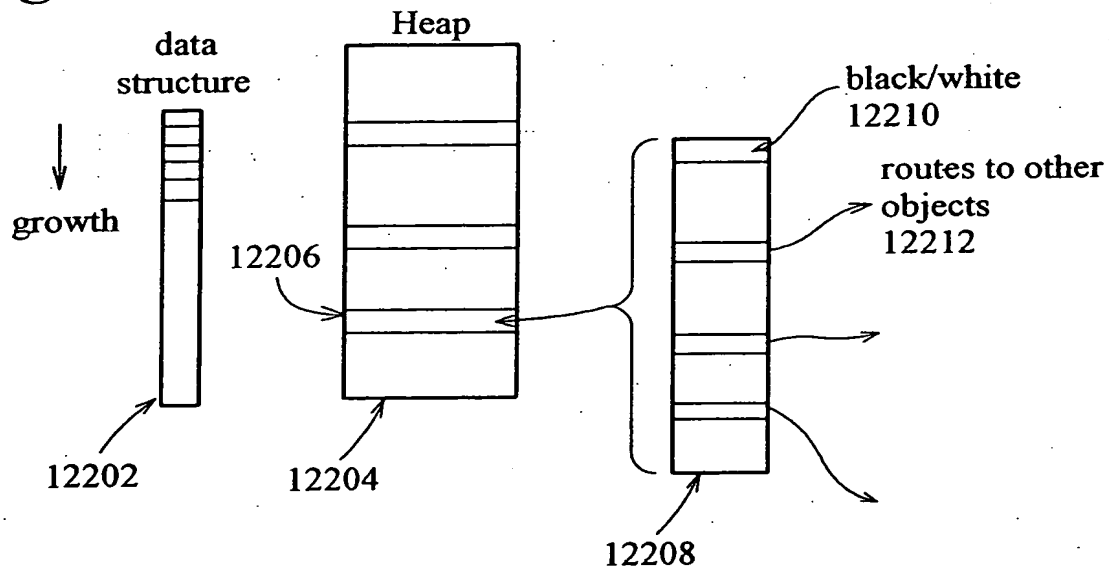
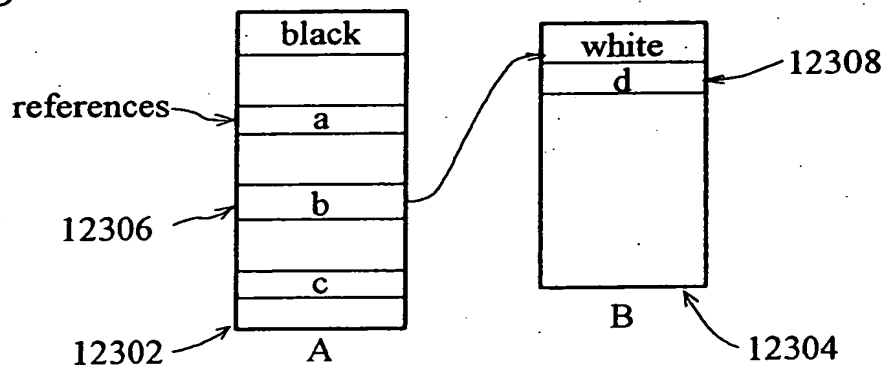
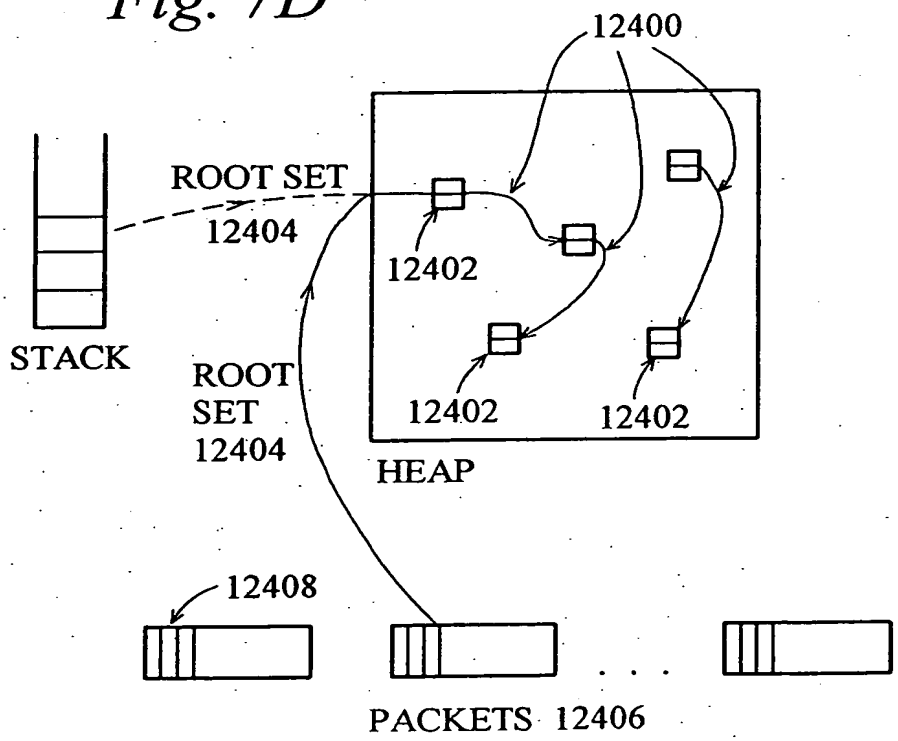


Fig. 7C



09538755 - 050704

Fig. 7D



08854250.050501

Fig. 7E

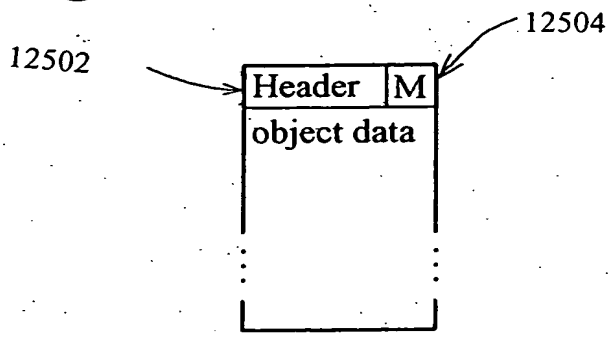


Fig. 7F

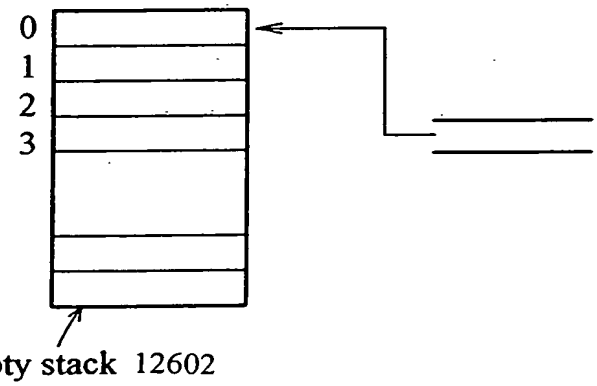


Fig. 7G

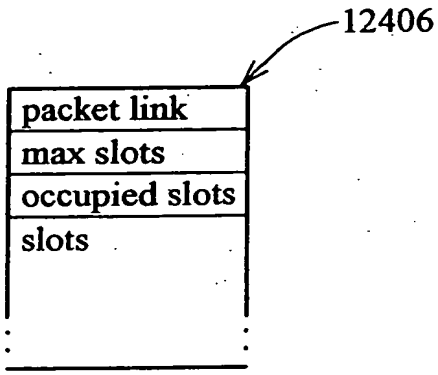
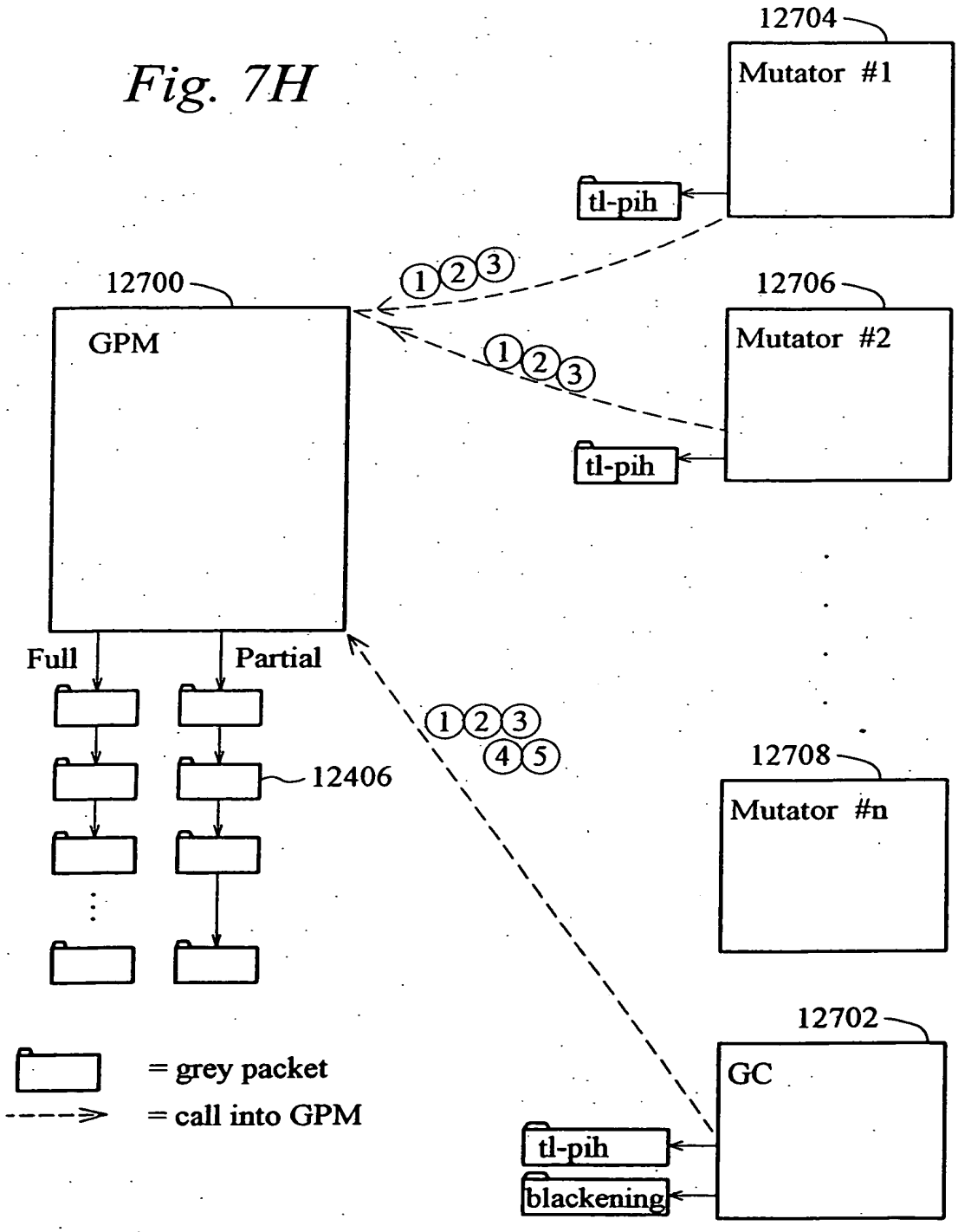


Fig. 7H


12704
Mutator #1
tl-pih
12700
GPM
1 2 3
12706
Mutator #2
1 2 3
tl-pih
Full
Partial
1 2 3
4 5
12406
12708
Mutator #n
12702
GC
= grey packet
= call into GPM
tl-pih
blackening

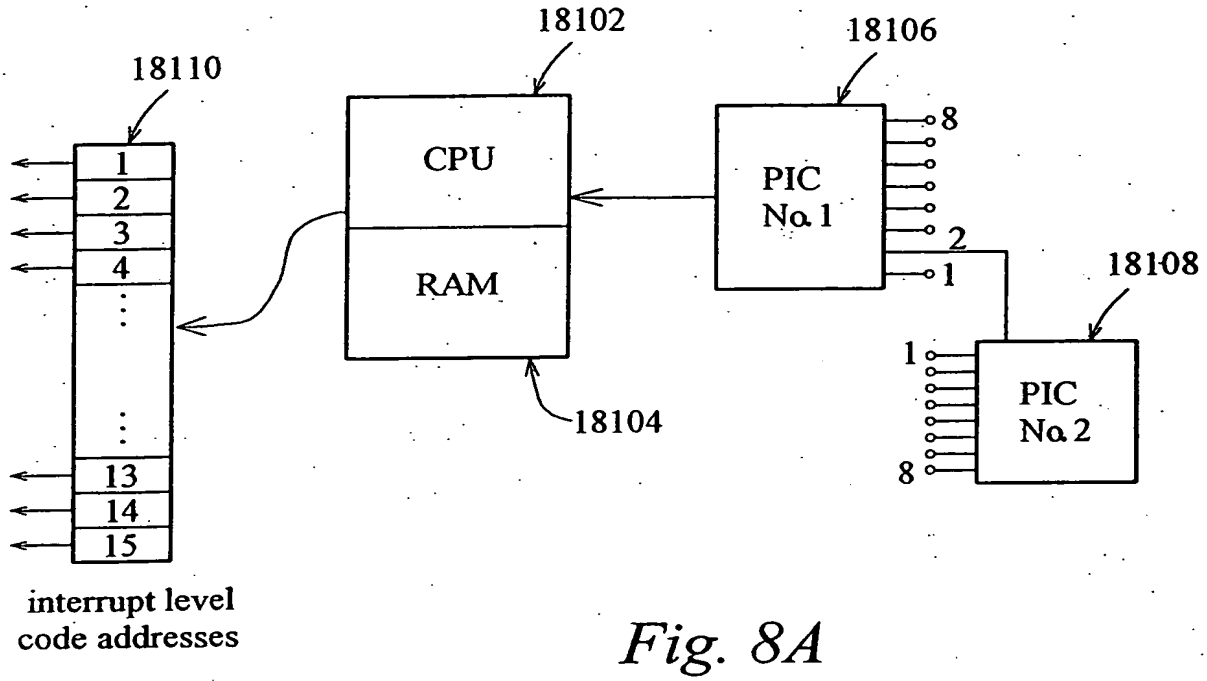


Fig. 8A

09854520.050501

Sequence of Events for Various System Components *Fig. 8B*

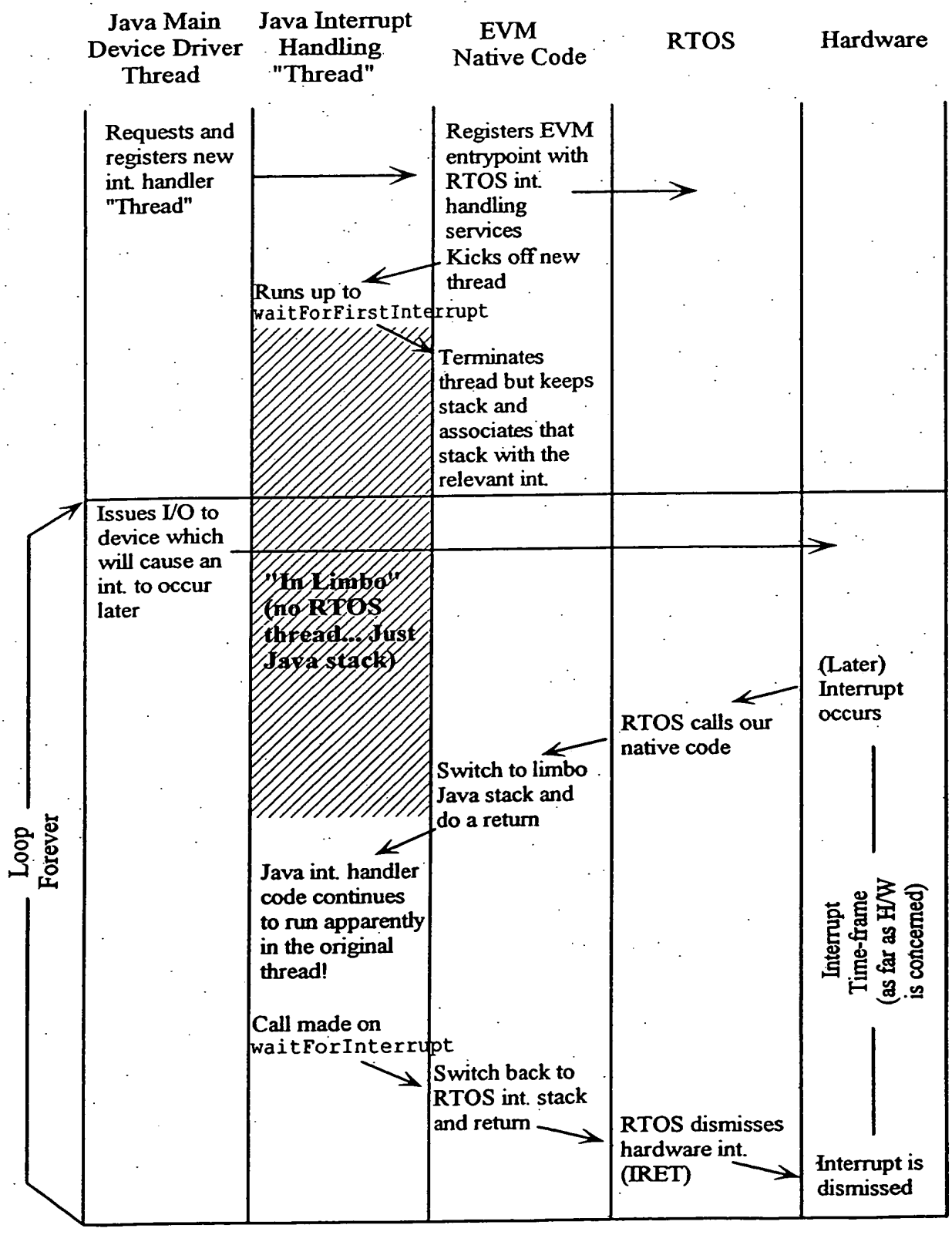


Fig. 8C-1

Pseudo-code of a Java Interrupt Handler

```
public void run ( )
{
        // The run method of an example interrupt handling thread

        // Wait for the first interrupt
        if  ( !waitForFirstInterrupt ( ) )
        {
                throw new RuntimeException  ( "Error waiting for 1st interrupt" ) ;
        }

        // We are now running at interrupt level!

        while  (true)
        {
                // Now handle the interrupt that just occurred (this involves
                // reading a device register)

                byte value = dev.readByte (DEV.DATA_REG);

                if  ( ( value & DEV.GOING_SYNC)  != 0)
                {
                        // We enter a sub-loop handling interrupts while in
                        // "synchronous mode"

                        boolean stillSync = true;
                        do
                        {
                                // Wait for the next interrupt to occur

                                waitForInterrupt ( ) ;

                                // Read the hardware data register

                                value = dev.readByte  (DEV.DATA_REG) ;
```

go to Fig. 8C-2

Fig. 8C-2

from Fig. 8C-1

```
                    // Decide if the value means that we are switching back
                    // to "async mode"

                    if ( (value & DEV.GOING_ASYNC) == 0)
                    {
                        // Handle "synchronous mode" interrupt here (just
                        // write the device data to non-interrupt code via
                        // the special channel)

                        specialChannel.write (value);
                    }
                    else
                    {
                        // Wait for the next interrupt to occur then return
                        // to the outer "async" loop

                        waitForInterrupt ( ) ;
                        stillSync = false;
                    }
                }
                while (stillSync)
            }

            // Handle "asynchronous mode" interrupts here (just write the
            // device data to non-interrupt code via the special channel)

            specialChannel.write  (value) ;

            waitForInterrupt ( ) ;
        }
    }
```

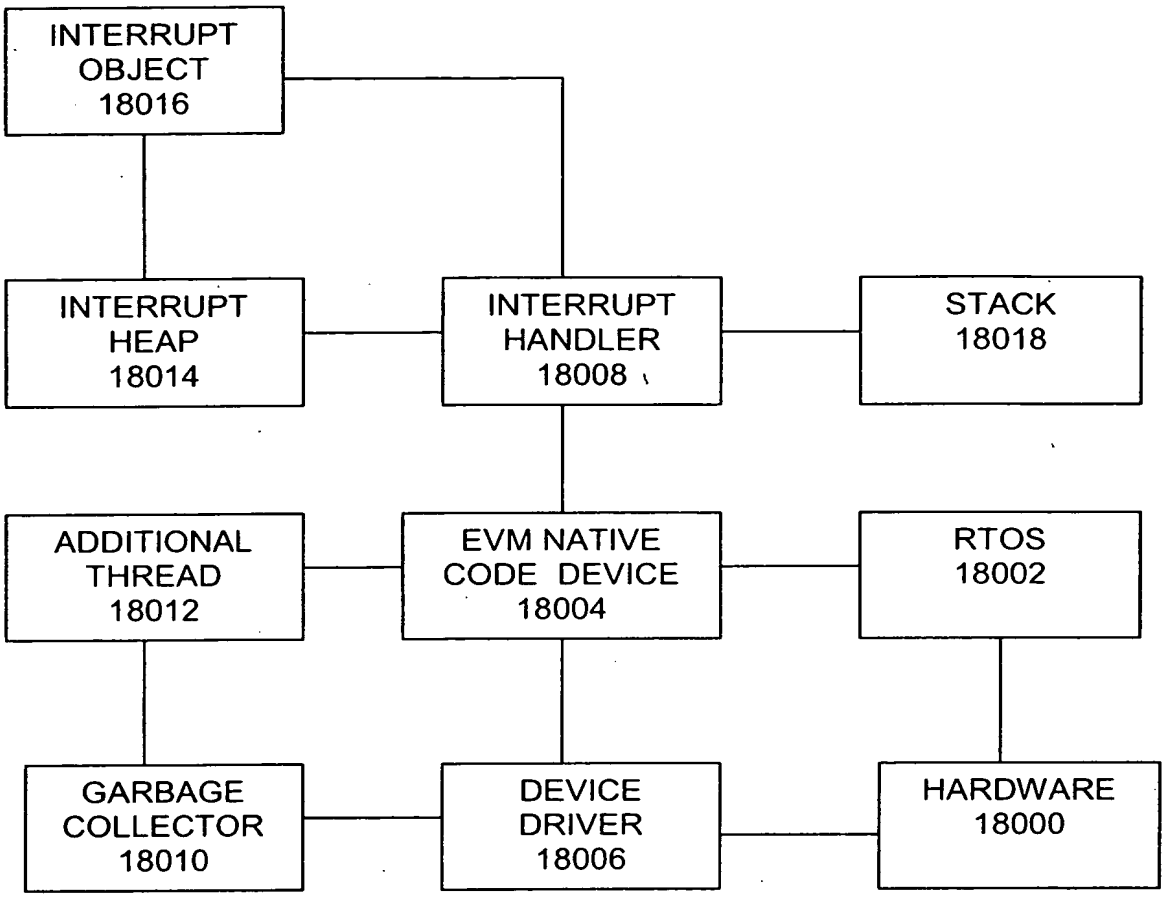
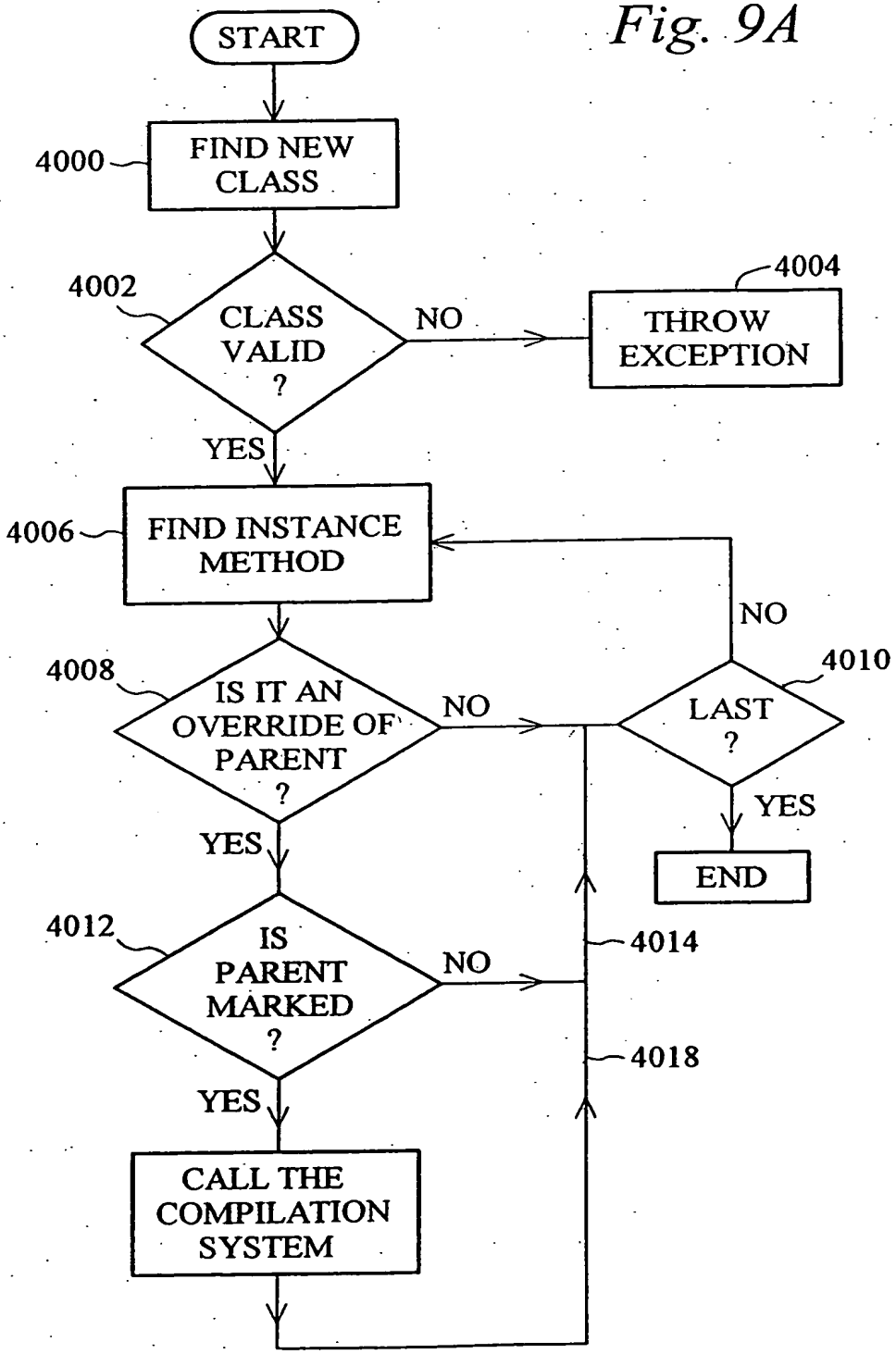

INTERRUPT OBJECT 18016
INTERRUPT HEAP 18014
INTERRUPT HANDLER 18008
STACK 18018
ADDITIONAL THREAD 18012
EVM NATIVE CODE DEVICE 18004
RTOS 18002
GARBAGE COLLECTOR 18010
DEVICE DRIVER 18006
HARDWARE 18000

FIG. 8D

Fig. 9A


START
4000
FIND NEW CLASS
4002
CLASS VALID ?
NO
4004
THROW EXCEPTION
YES
4006
FIND INSTANCE METHOD
NO
4008
IS IT AN OVERRIDE OF PARENT ?
NO
4010
LAST ?
YES
END
YES
4012
IS PARENT MARKED ?
NO
4014
4018
YES
CALL THE COMPILATION SYSTEM

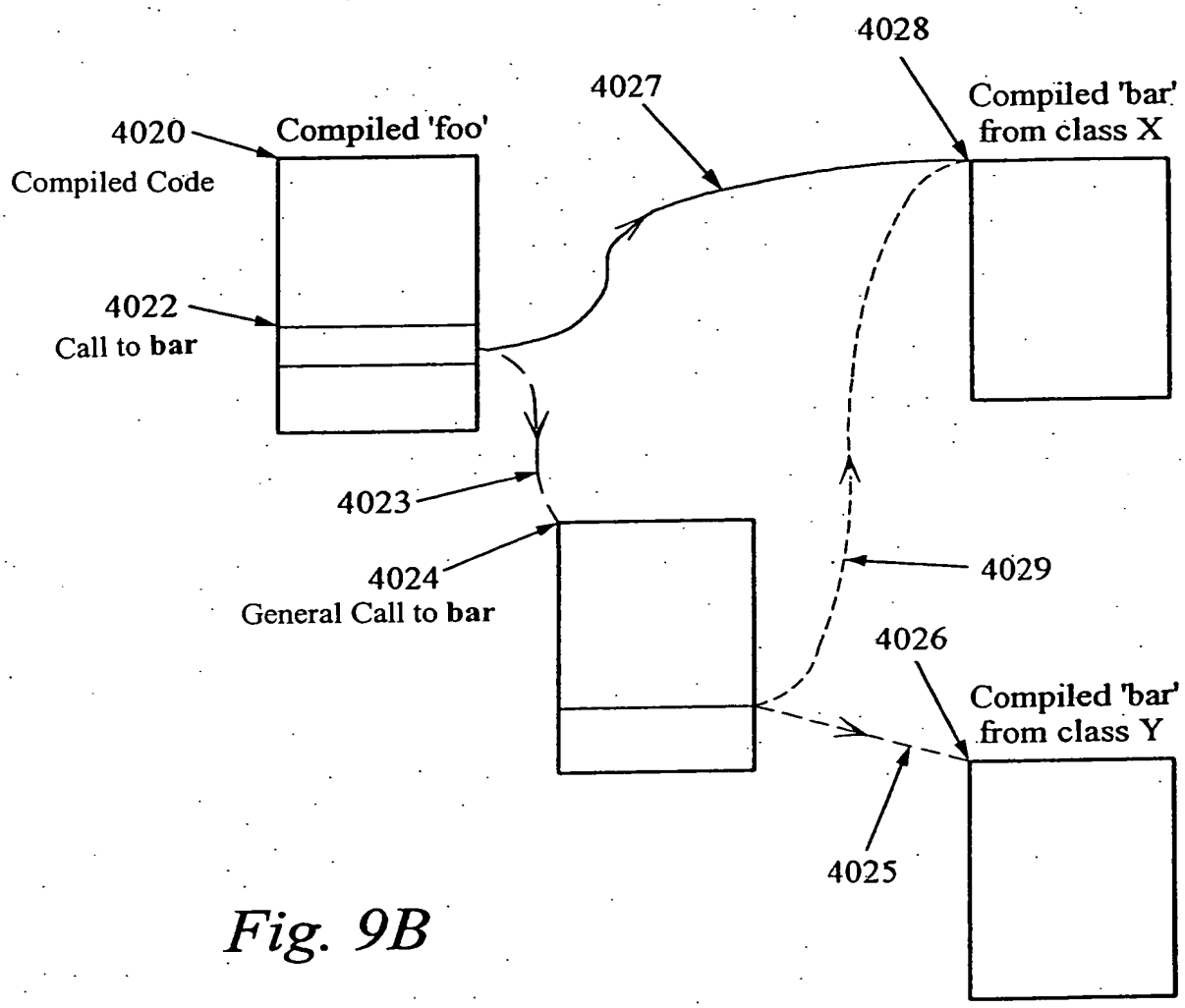


Fig. 9B

09805420.050501

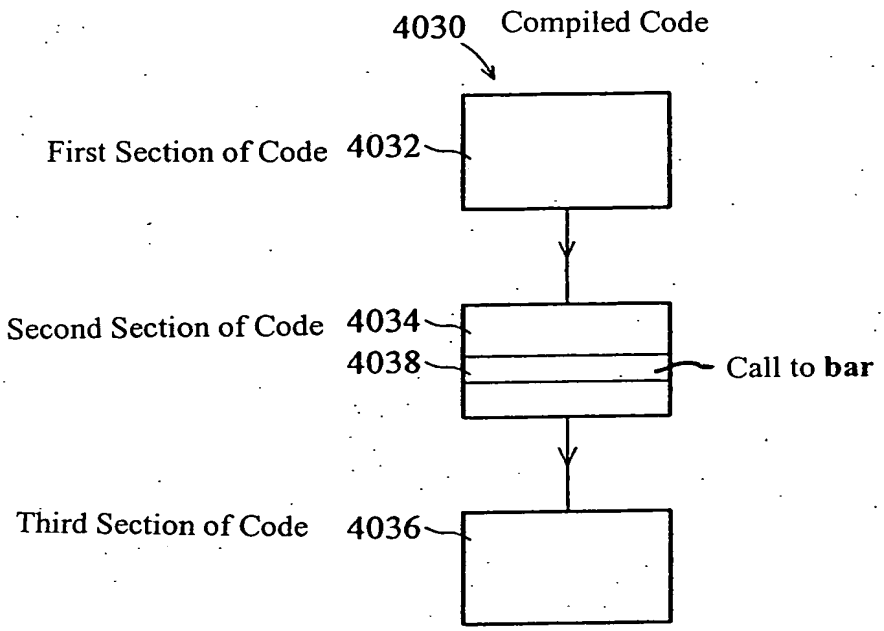


Fig. 9C

09853456.050501

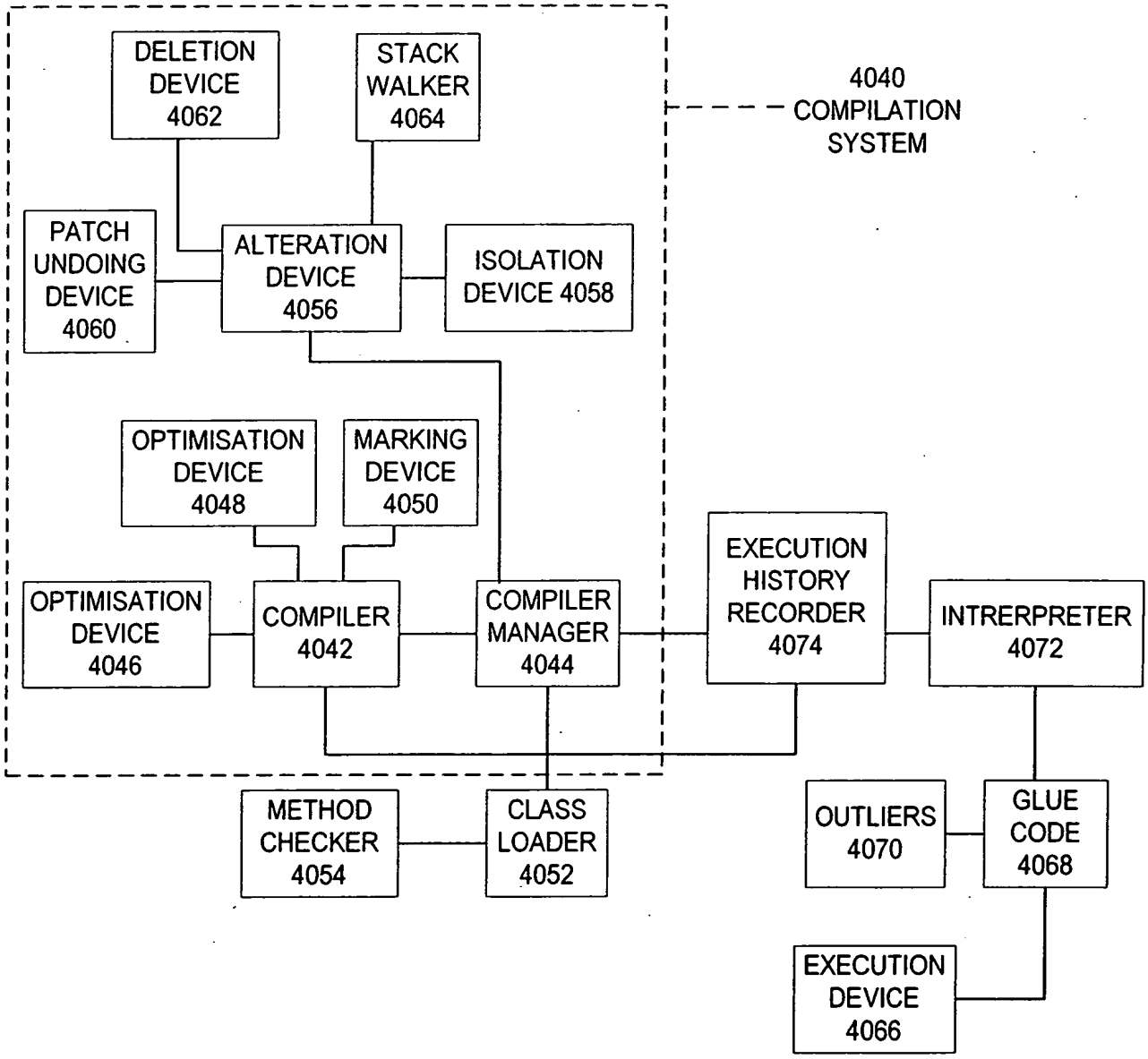

DELETION DEVICE 4062
STACK WALKER 4064
4040 COMPILATION SYSTEM
PATCH UNDOING DEVICE 4060
ALTERATION DEVICE 4056
ISOLATION DEVICE 4058
OPTIMISATION DEVICE 4048
MARKING DEVICE 4050
OPTIMISATION DEVICE 4046
COMPILER 4042
COMPILER MANAGER 4044
EXECUTION HISTORY RECORDER 4074
INTRERPRETER 4072
METHOD CHECKER 4054
CLASS LOADER 4052
OUTLIERS 4070
GLUE CODE 4068
EXECUTION DEVICE 4066

FIG. 9D

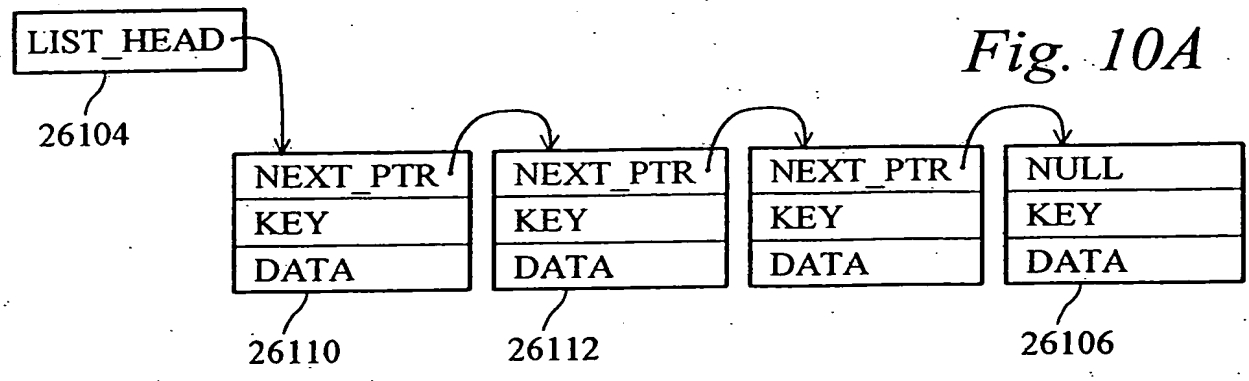


Fig. 10A

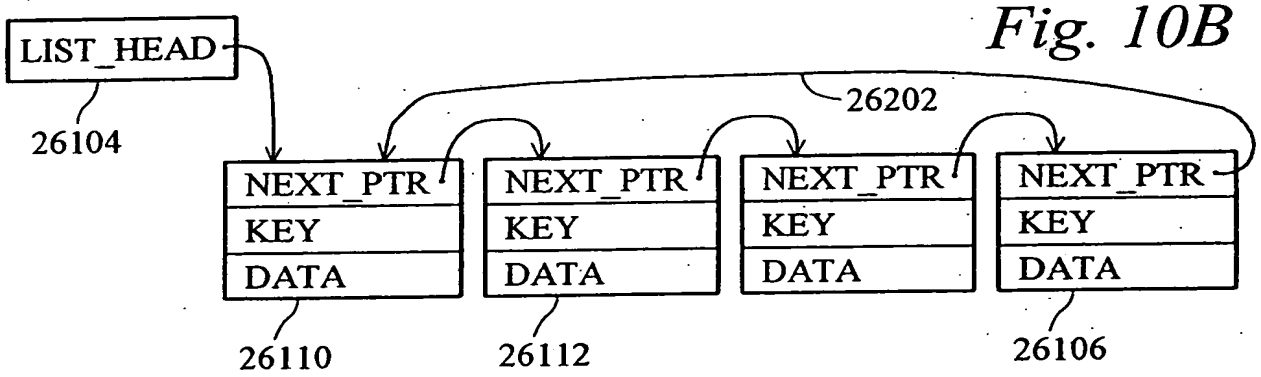


Fig. 10B

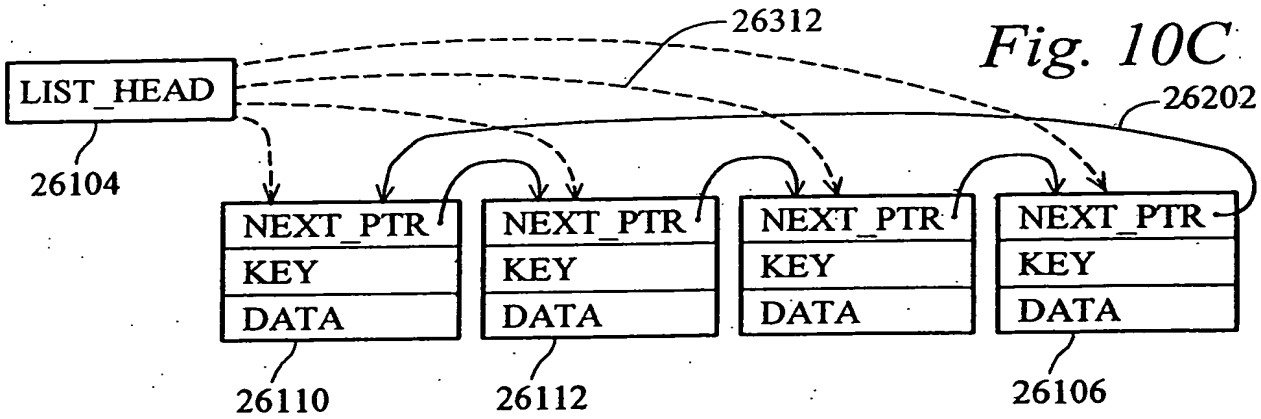


Fig. 10C

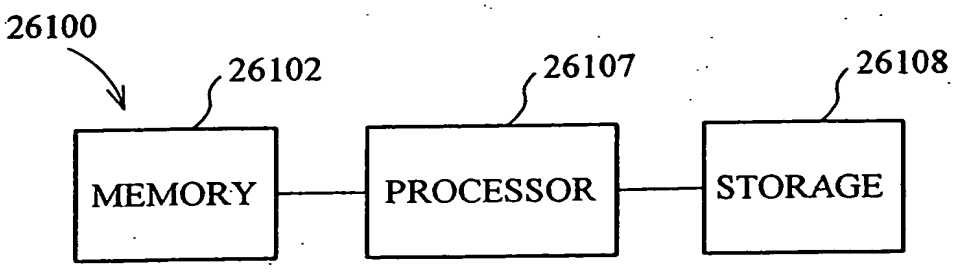


Fig. 10D

Fig. 11A

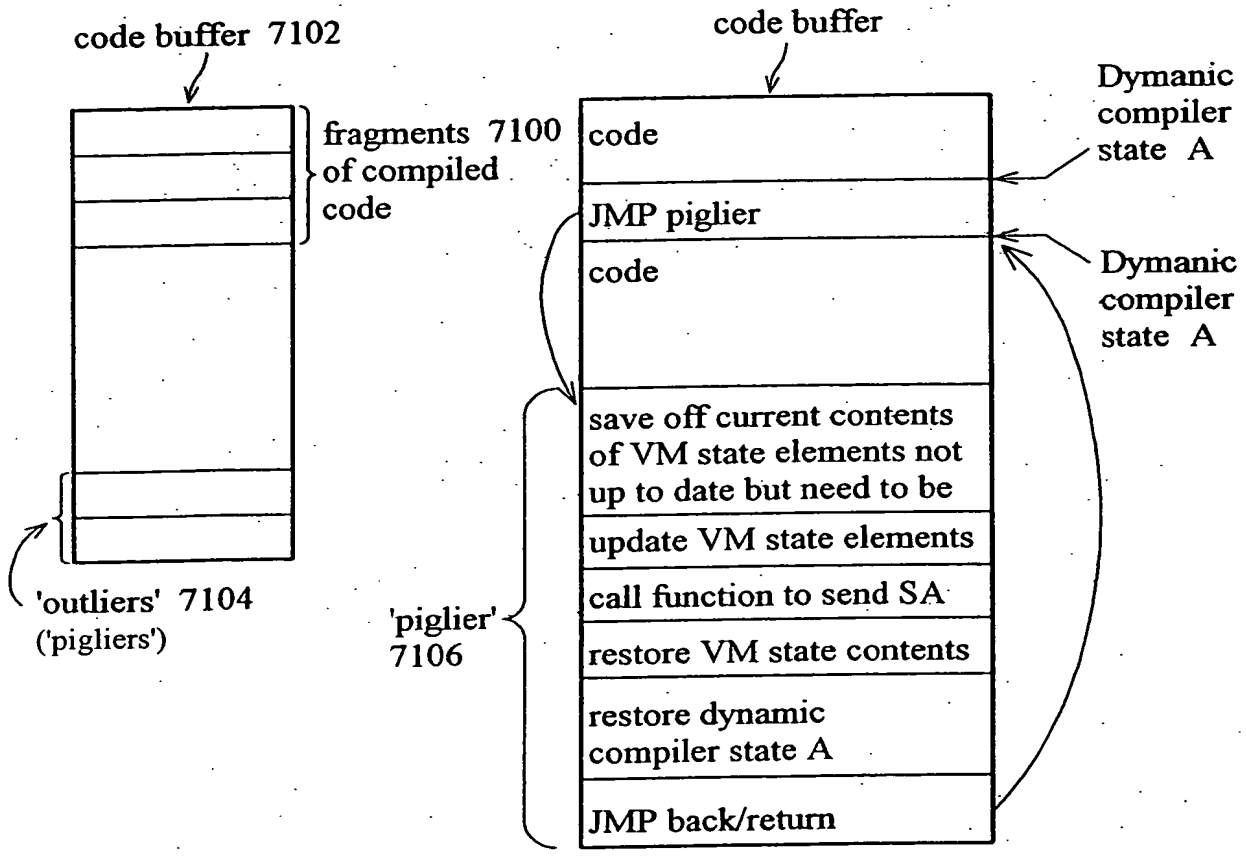
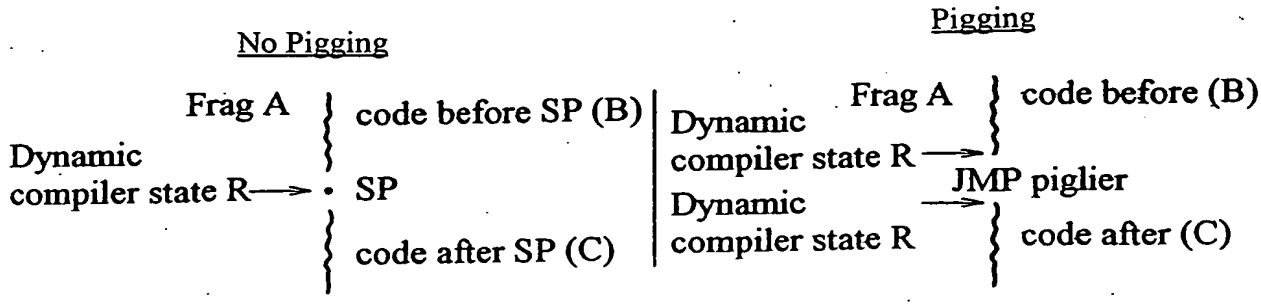


Fig. 11B



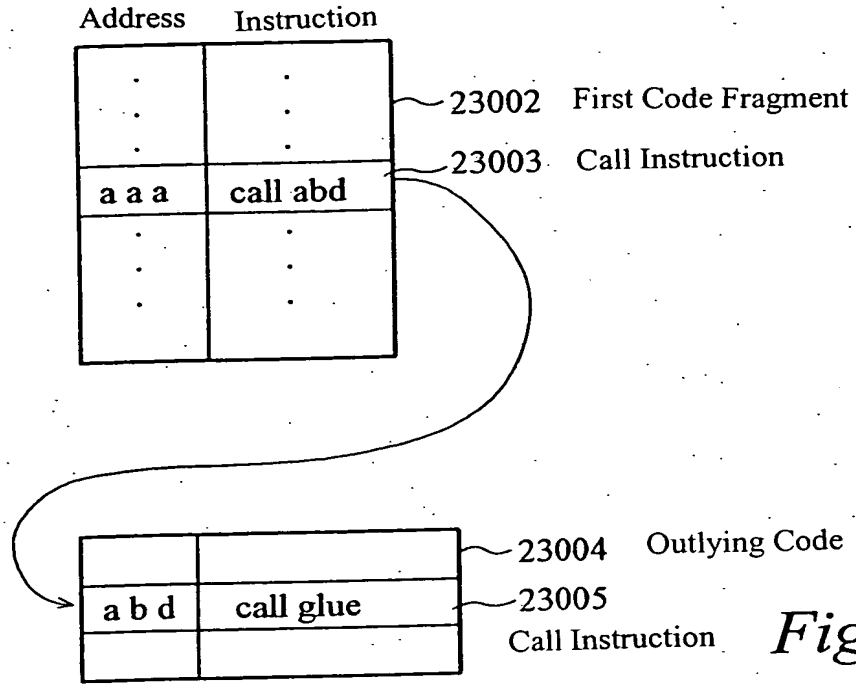


Fig. 12A

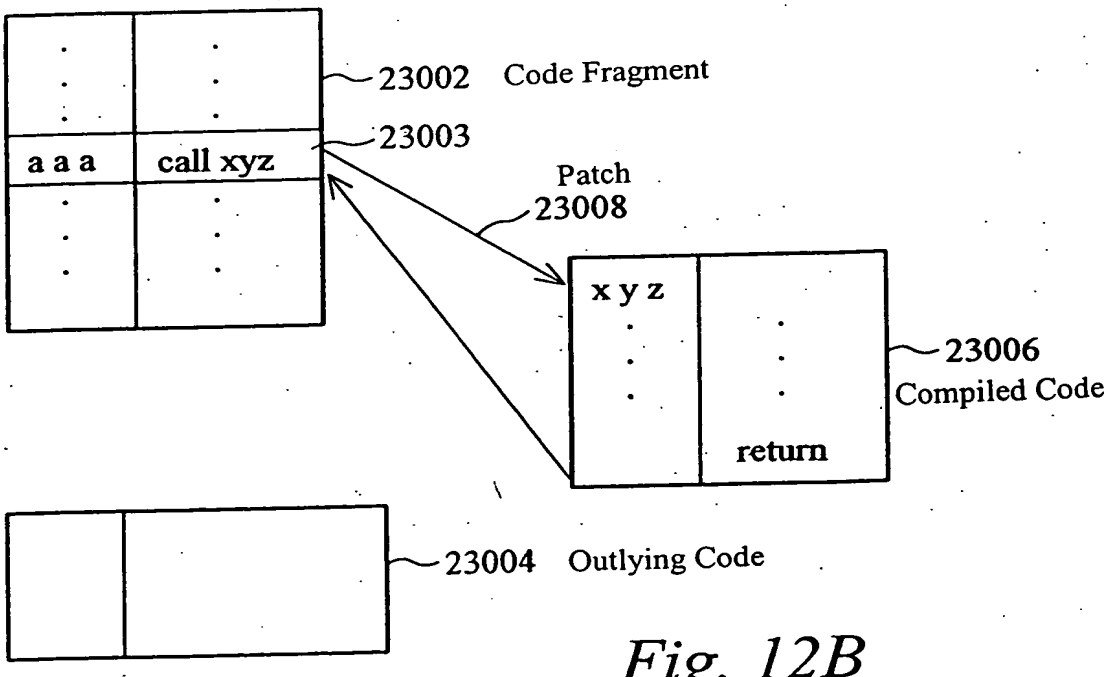


Fig. 12B

Docket No. 50325-0150

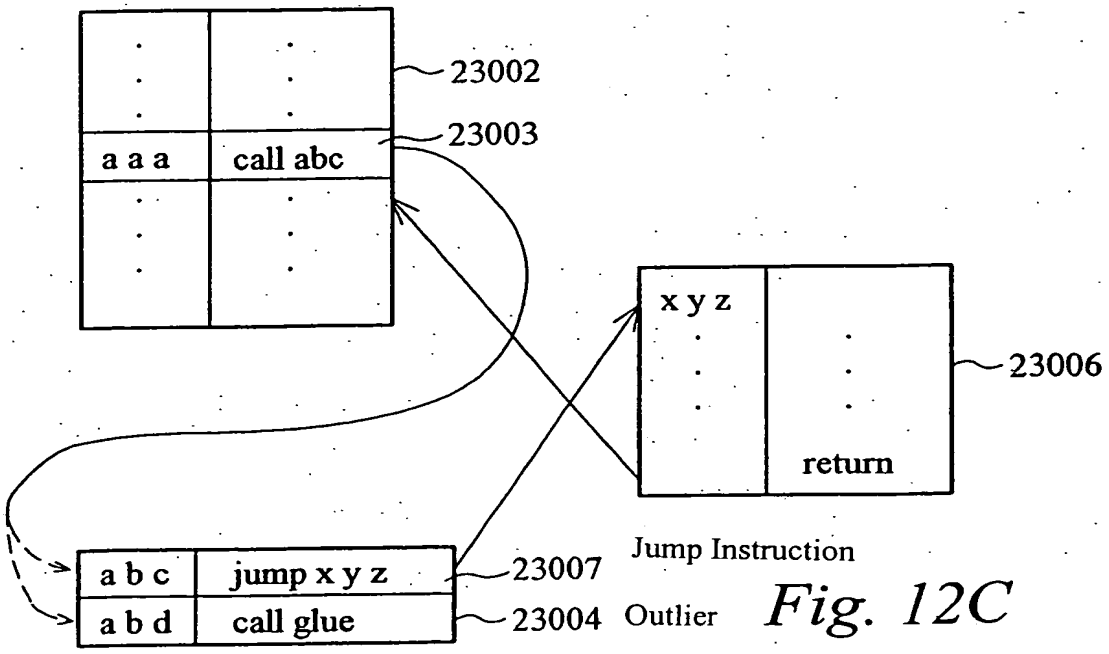


Fig. 12C

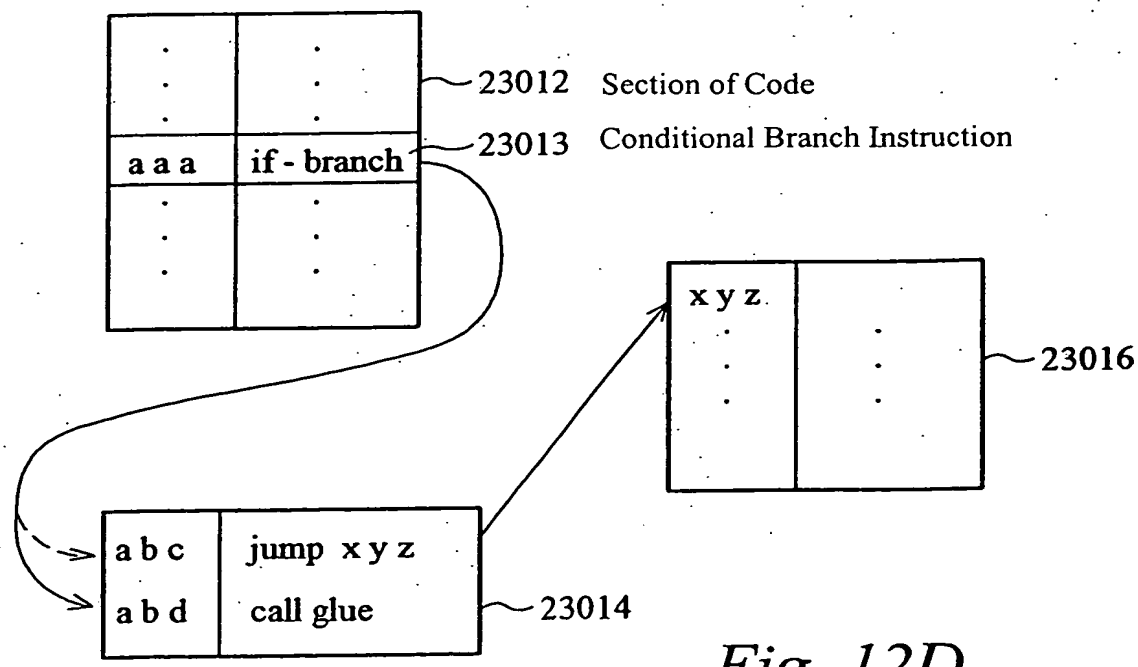


Fig. 12D

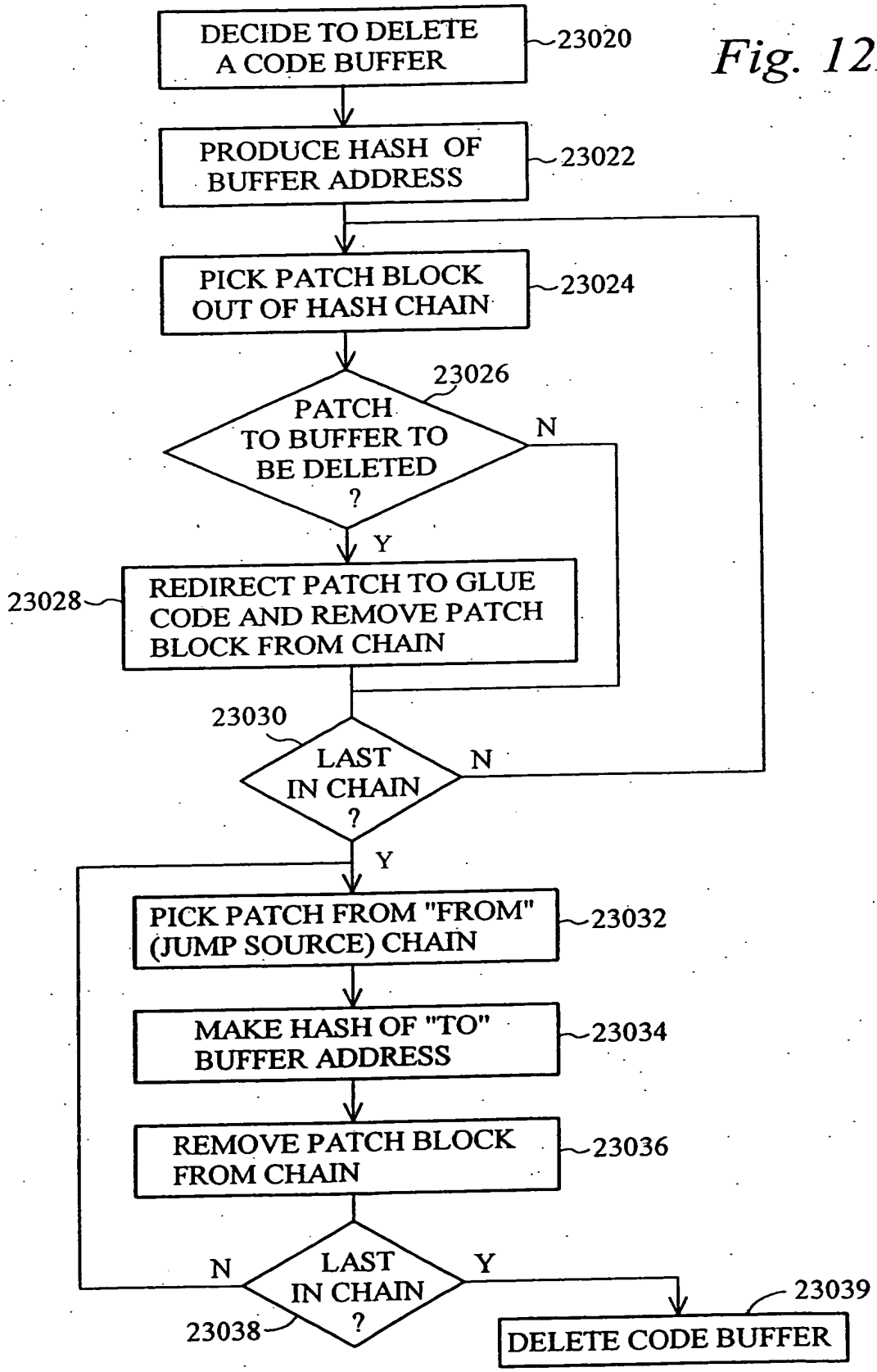

DECIDE TO DELETE A CODE BUFFER
23020
PRODUCE HASH OF BUFFER ADDRESS
23022
PICK PATCH BLOCK OUT OF HASH CHAIN
23024
23026
PATCH TO BUFFER TO BE DELETED ?
N
Y
23028
REDIRECT PATCH TO GLUE CODE AND REMOVE PATCH BLOCK FROM CHAIN
23030
LAST IN CHAIN ?
N
Y
PICK PATCH FROM "FROM" (JUMP SOURCE) CHAIN
23032
MAKE HASH OF "TO" BUFFER ADDRESS
23034
REMOVE PATCH BLOCK FROM CHAIN
23036
LAST IN CHAIN ?
N
Y
23038
23039
DELETE CODE BUFFER

Fig. 12E

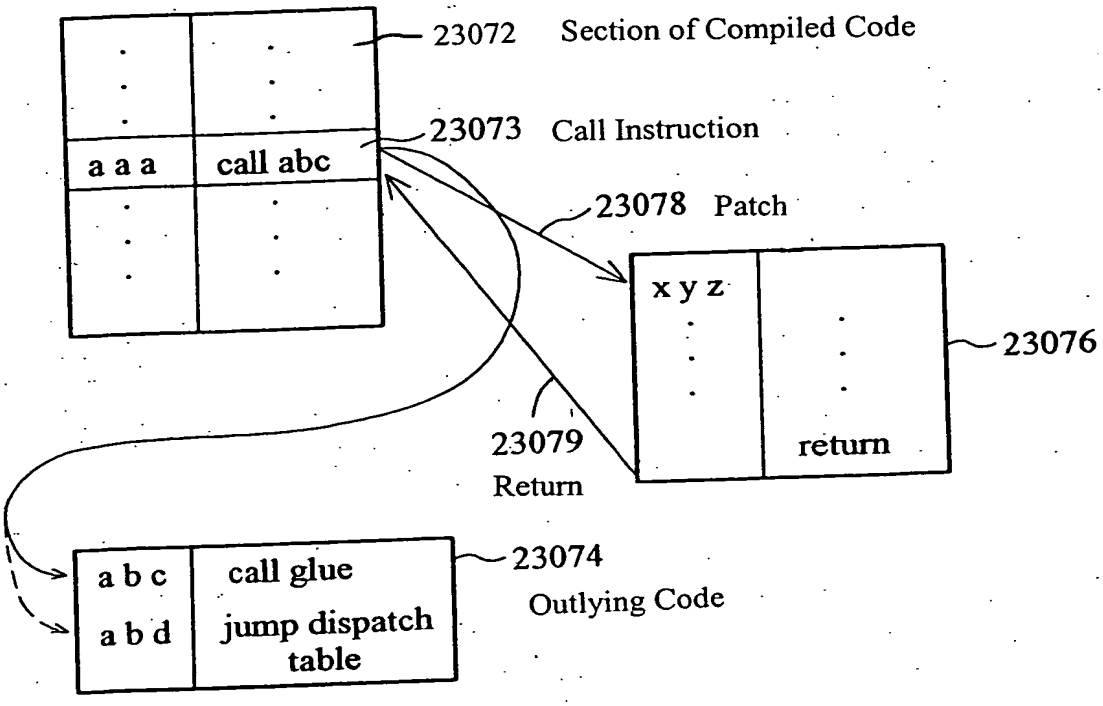


Fig. 12F

d08584n0 - 05450p

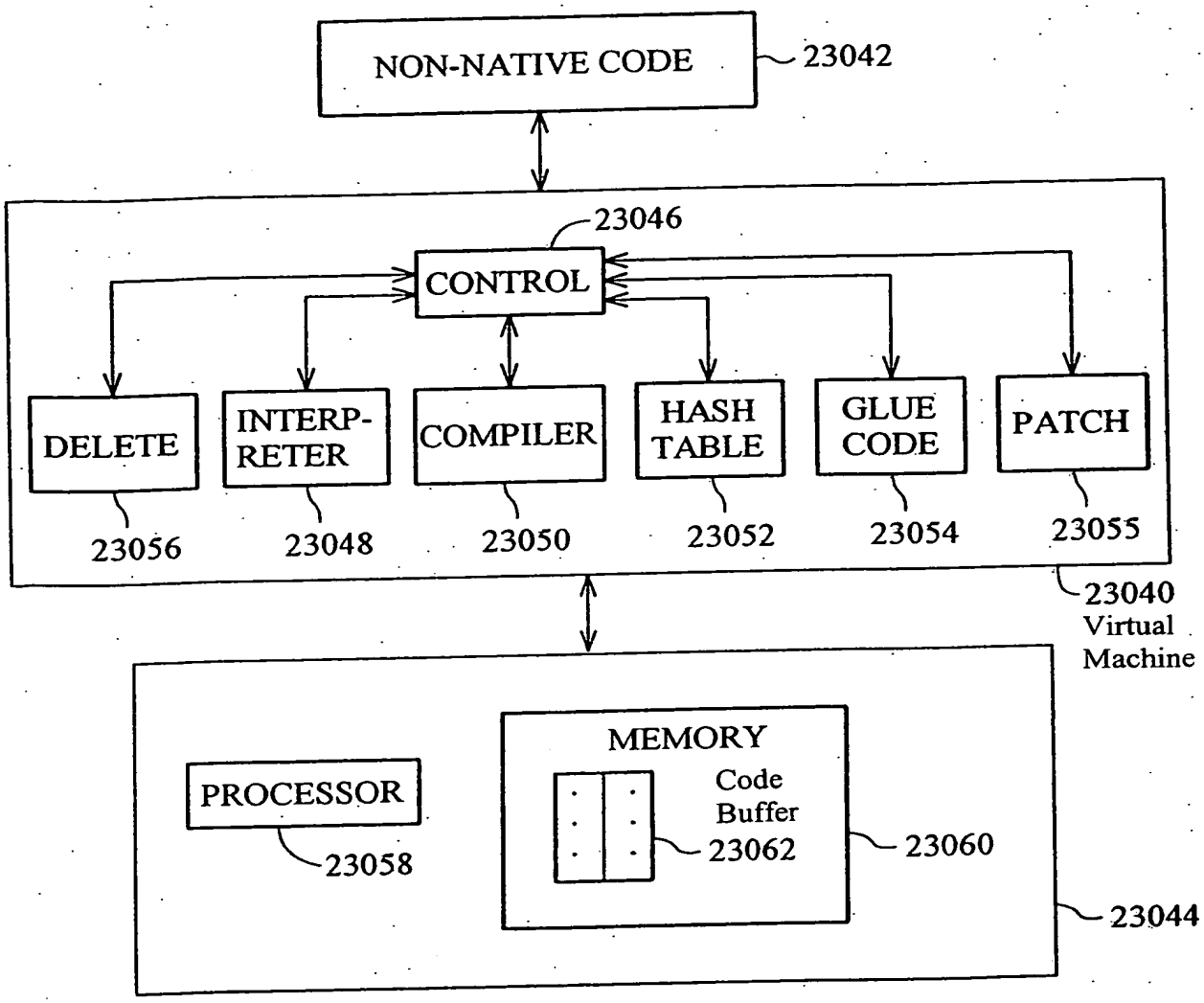

NON-NATIVE CODE
23042
23046
CONTROL
DELETE
INTERP-RETER
COMPILER
HASH TABLE
GLUE CODE
PATCH
23056
23048
23050
23052
23054
23055
23040
Virtual Machine
PROCESSOR
23058
MEMORY
Code Buffer
23062
23060
23044

Fig. 12G